مرکز آموزش عالی محلات

دانشکده فنی و مهندسی، گروه کامپیوتر

پایان نامه برای دریافت درجه کارشناسی

گرایش: نرم‌افزار

عنوان:

جنگ جهان‌ها: یک بازی استراتژی زمان‌واقعی آنلاین

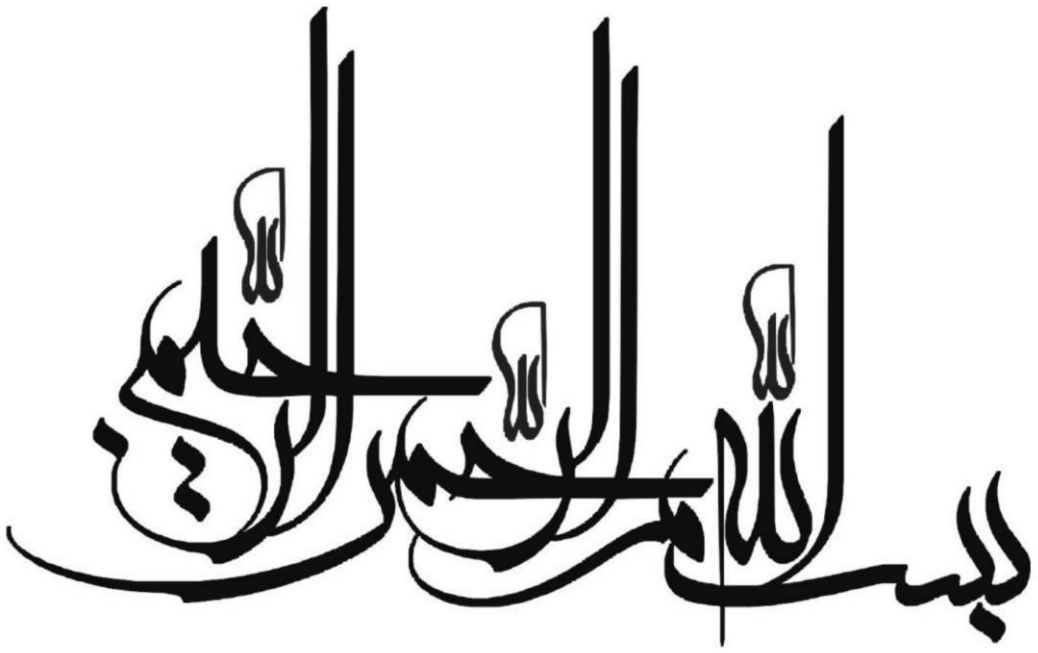
استاد راهنما:

دکتر سعید دوستعلی

نگارش:

نیما ستاری

زمستان ۱۴۰۰


بسم الله الرحمن الرحيم

فرم تعهدنامه‌ی حق چاپ و تکثیر و مالکیت نتایج

احتراماً اینجانبان (امضا کنندگان ذیل)، متعهد می‌شویم که مطالب و نتایج تحقیقاتی که حاصل از پایان‌نامه دانشجو نیما ستاری با عنوان : " ساخت یک بازی استراتژی زمان واقعی" که در مرکز آموزش عالی محلات به تصویب رسیده است، چنانچه به صورت مقاله، اختراع، کتاب و ... منتشر شود، نام مرکز آموزش عالی محلات در کنار نویسندگان و ارایه‌کنندگان این تولیدات علمی، به نحوی که تعلق آن اثر به مرکز آموزش عالی محلات را کاملاً مسجل نماید، ذکر گردد و در صورت عدم رعایت این موارد عواقب ناشی از آن را پذیرفته و واحد دانشگاهی مجاز است مطابق با مقررات و ضوابط برخورد نماید.

استاد راهنما نام و نام خانوادگی: دکتر سعید دوستعلی تاریخ و امضا: ۱۴۰۰/۱۱/۰۵

دانشجو نام و نام خانوادگی: نیما ستاری تاریخ و امضا: ۱۴۰۰/۱۱/۰۵

تاریخ: ۱٤۰۰/۱۱/۰۵

شماره:

پیوست:

تعهدنامه اصالت پایان نامه

اینجانب نیما ستاری دانش آموخته مقطع کارشناسی در رشته مهندسی نرم افزار که در تاریخ ۱٤۰۰/۱۱/۰۵ از پایان نامه خود تحت عنوان: ساخت یک بازی استراتژی زمان واقعی با کسب نمره ..... و درجه ..... دفاع نموده‌ام، بدینوسیله متعهد می‌شوم:

۱) این پایان نامه حاصل تحقیق و پژوهش انجام شده توسط اینجانب بوده و در مواردی که از دستاوردهای علمی و پژوهشی دیگران (اعم از پایان نامه، کتاب ، مقاله و ...) استفاده نموده‌ام، مطابق ضوابط و رویه موجود، نام منبع مورد استفاده و سایر مشخصات آن را در فهرست مربوطه ذکر و درج کرده‌ام.

۲) این پایان نامه پیش از این برای دریافت هیچ مدرک تحصیلی (هم سطح، پایین‌تر یا بالاتر) در سایر دانشگاه ها و موسسات آموزش عالی ارائه نشده است.

۳) چنانچه بعد از فراغت از تحصیل، قصد استفاده و هرگونه بهره برداری اعم از چاپ کتاب، ثبت اختراع و... از این پایان نامه داشته باشم، از حوزه معاونت پژوهشی دانشگاه مجوزهای مربوطه را اخذ نمایم.

٤) چنانچه در هر مقطعی زمانی خلاف موارد فوق ثابت شود ، عواقاب ناشی از آن را میپذیرم و دانشگاه مجاز است با اینجانب مطابق ضوابط و مقررات رفتار نموده و درصورت ابطال مدرک تحصیلی‌ام هیچگونه ادعایی نخواهم داشت.

نام و نام خانوادگی: نیما ستاری

تاریخ و امضا: ۱٤۰۰/۱۱/۰۵

تقدیم

تقدیم به استاد گرامی جناب آقای مهندس سعید دوستعلی استاد فرزانه و فرهیخته‌ای که در راه کسب علم و معرفت مرا یازی نمودند.

سپاس‌گزاری

از خانواده خود بسیار سپاس‌گزارم زیرا بدون حمایت و پشتیبانی آنان تامین این پایان‌نامه بسیار دشوار می‌نمود و تشکر فراوان از زحمات شما در طول سالیان.

چکیده فارسی

جنگ جهان‌ها: یک بازی استراتژی زمان‌واقعی آنلاین

استراتژی زمان واقعی یک زیر گروه از بازی‌های ویدئویی استراتژی است که بازیکنان به صورت نوبتی بازی نمی‌کنند و در واقع به همه بازیکنان اجازه می‌دهد به طور همزمان ، در "زمان واقعی" بازی کنند. در کشور ما، ایران با توجه به بازار بزرگ و گسترده‌ای که بازی‌های رایانه‌ای دارند و بازی‌های استراتژی که جزء پرطرفدار-ترین ژانرها می‌باشد پس لازم است تا از این بازار استفاده کنیم. ابتدا به توضیح استراتژی زمان واقعی و تاریخچه آن می‌پردازیم و سپس چندین نمونه ساخته شده توسط شرکت‌های بزرگ را توضیح داده و سپس به مقایسه آنان می‌پردازیم. در این پروژه یک بازی استراتژی زمان واقعی را تحلیل می‌کنیم سپس نمودارهای آن را ارائه می‌دهیم و پس از آن به کمک موتور بازی سازی یونیتی که یکی از معروف‌ترین برنامه‌های توسعه بازی است، زبان برنامه سازی سی‌شارپ و برنامه Mirror که برای اتصال بازی‌ها به شبکه در یونیتی به کار می‌رود، این پروژه را پیاده سازی می‌کنیم. این پروژه در یک محیط فانتزی-قرون وسطایی نمایش داده می‌شود. ابتدا سیستم مسیریابی را به کمک AI Navigation یونیتی پیاده‌سازی میکنیم، سپس سیستم انتخاب و فرمان به نیروها را به وسیله Physics.Raycast ساختارسازی میکنیم. در ادامه سیستم تولید و نابودی متعلقات بازی(ساختمان‌ها و نیروها) را به کمک Eventها و Mirror می‌سازیم. در پایان قابلیت آنلاین چندنفره را به وسیله‌ی FizzySteamworks پیاده‌سازی می‌کنیم.

ساخت ساختمان‌های تعلیم نیرو و انبار منابع، تعلیم نیروهای کارگر و سرباز، استخراج منابع، حرکت و حمله توسط نیروها، برد و باخت، نقشه کوچک و منو اصلی اشاره کرد. همچنین قابلیت آنلاین آن که از ۲ تا ٤ بازیکن به صورت همزمان میتوانند با یکدیگر بازی کنند و به رقابت بپردازند که این قابلیت در نرم‌افزار Steam قابل دریافت است.

کلمات کلیدی: یونیتی، Mirror، استراتژی زمان واقعی، برنامه نویسی #C، برنامه‌نویسی شی‌گرا

فهرست مطالب

فصل اول

کلیات

۱-۱- مقدمه

استراتژی زمان واقعی [1] یک زیرگروه از بازی‌های ویدئویی استراتژی است که بازیکنان به صورت نوبتی بازی نمی‌کنند و درواقع به همه بازیکنان اجازه می‌دهد به طور همزمان ، در "زمان واقعی" بازی کنند. در مقابل ، در بازی‌های استراتژی مبتنی بر نوبت [2]، بازیکنان به نوبت به بازی می‌پردازند. اصطلاح "استراتژی زمان واقعی" توسط برت اسپری برای عرضه Dune II در اوایل دهه ۱۹۹۰ مطرح شد.[۱]

در یک بازی استراتژی زمان واقعی ، هر یک از بازیکنان چندین واحد را تحت کنترل غیرمستقیم خود قرار می‌دهند و برای ایمن سازی مناطق نقشه و/یا از بین بردن دارایی‌های رقیبان خود مانور می‌دهند. در یک بازی معمولی استراتژی زمان واقعی ، ممکن است واحدها و ساختارهای اضافی ایجاد شود ، که عموماً با نیاز به صرف منابع انباشته محدود می‌شوند. این منابع به نوبه خود با کنترل نقاط ویژه روی نقشه و/یا دارا بودن انواع خاصی از واحدها و ساختارهای اختصاص داده شده به این هدف ، جمع آوری می‌شوند. به طور خاص ، بازی معمولی در ژانر استراتژی زمان واقعی شامل جمع آوری منابع ، ایجاد پایگاه ، توسعه تکنولوژیکی درون بازی و کنترل غیر مستقیم واحدها است.[۱،۲]

وظایفی که بازیکن باید برای برنده شدن در یک بازی استراتژی زمان واقعی انجام دهد می‌تواند بسیار سخت باشد و رابط‌های کاربری پیچیده‌ای برای آن‌ها ایجاد شده است. برخی از ویژگی‌ها از محیط‌های رومیزی وام گرفته شده است. به عنوان مثال ، تکنیک "کلیک کردن و کشیدن" برای ایجاد یک مستطیل که همه واحدها را در یک منطقه مشخص انتخاب می‌کند. اگرچه برخی از ژانرهای بازی‌های ویدئویی شباهت‌های ظاهری و گیم‌پلی با الگوی استراتژی زمان واقعی دارند ، اما ژانرهای شناخته شده عموماً به عنوان بازی‌های استراتژی زمان واقعی شناخته نمی‌شوند. به عنوان مثال ، بازی‌های شهرسازی ، شبیه‌سازی‌های ساخت و مدیریت و بازی‌های تاکتیک‌های زمان واقعی عموماً به خودی خود استراتژی زمان واقعی در نظر گرفته نمی‌شوند. این فقط در مورد هر چیزی که به عنوان بازی خدا[3] در نظر گرفته شود صدق می‌کند ، جایی که بازیکن نقش خلقت خداگونه را بر عهده می‌گیرد.[۱،۲]

[1] Real-Time Strategy (RTS)
[2] Turn-Based Strategy (TBS)
[3] God Mode Game

۱-۲- منشا

ژانری که امروزه به عنوان "استراتژی زمان واقعی" شناخته می‌شود ، از یک دوره طولانی تکامل و پالایش پدید آمده است. بازی‌هایی که بعضاً به عنوان اجداد ژانر استراتژی زمان واقعی تلقی می‌شدند ، هرگز به بازار عرضه نشدند و به این صورت طراحی نشدند. در نتیجه ، تعیین عناوین "استراتژی اولیه در زمان واقعی" مشکل ساز است زیرا چنین بازی‌هایی با استانداردهای مدرن برگزار می‌شوند. این ژانر در ابتدا به طور جداگانه در انگلستان ، ژاپن و آمریکای شمالی تکامل یافت و سپس به تدریج در یک سنت جهانی واحد ادغام شد.

تیم بری در ماه مه ۱۹۸۱ در InfoWorld یک بازی فضایی استراتژی چند نفره و زمان واقعی را توصیف کرد که بر روی سیستم IBM System/370 Model 168 در یک شرکت بزرگ منطقه خلیج سانفرانسیسکو اجرا شد. بری با مقایسه پیچیدگی آن با دالاس ، به یاد می‌آورد که "وقتی بازی در ساعت ۵ بعد از ظهر بازیابی شد ، بسیاری از کارهای معمولی متوقف شد".[۳]

مجله Ars Technica ریشه های این ژانر را در Utopia (۱۹۸۱) می‌داند و از آن به عنوان "تولد یک ژانر" یاد می‌کند ، با "عنصر در زمان واقعی" که عملاً شنیده نشده بود، بنابراین "مسلماً اولین نیاکان ژانر استراتژی زمان واقعی" است.[٤] به گفته Ars Technica، Utopia یک بازی استراتژی نوبتی با عناصر ترکیبی بود که "در زمان واقعی رخ میداد اما رویدادها در یک چرخه معمولی مبتنی بر نوبت اتفاق می‌افتاد."[۵] به گفته برت وایس ، Utopia اغلب به عنوان "اولین بازی استراتژی زمان واقعی" ذکر می‌شود.[٦] به گفته مت بارتون و بیل لوگوئدیس ، Utopia "در تنظیم الگو" برای این ژانر کمک کرد ، اما "شباهت بیشتری با SimCity نسبت به بازی Dune II و بازی های بعدی استراتژی زمان واقعی دارد"[۷].مجله Allgame بازی War of Nerves را به عنوان قدیمی‌ترین "استراتژی ۲ بعدی در زمان واقعی" ذکر کرده است.[۸] همچنین بارتون به Cytron Masters (۱۹۸۲) اشاره می‌کند و می‌گوید "این یکی از اولین بازی های استراتژی زمان واقعی بود."[۹] از سوی دیگر ، اسکات شارکی از UP۱ استدلال می‌کند که در حالی که Cytron Masters "سعی در بودن به استراتژی زمان واقعی" داشت ، به دلیل "ناتوانی در ایجاد واحدها یا مدیریت منابع" "بیشتر تاکتیکی محسوب می‌شود تا استراتژیک".[۱۰] شرکت BYTE در دسامبر ۱۹۸۲ Cosmic Conquest را برای Apple 2 منتشر شد. برنده مسابقه مجله سالانه بازی ،نویسنده آن را به عنوان "بازی تک نفره عمل در زمان واقعی و تصمیم‌گیری استراتژیک" توصیف کرد. مجله آن

را "یک بازی استراتژی فضایی در زمان واقعی" توصیف کرد. این بازی دارای عناصر مدیریت منابع و جنگ بازی است.[۱۱]

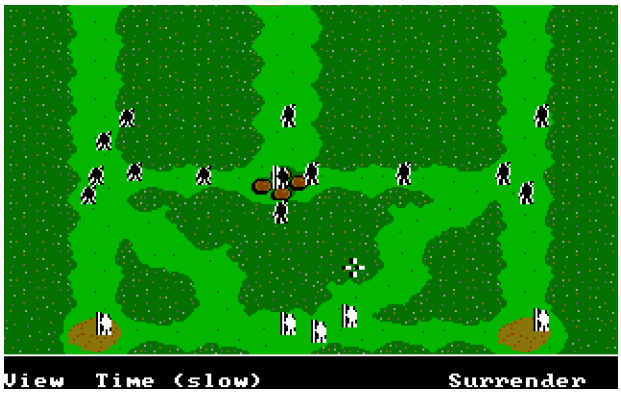
در انگلستان ، اولین بازی های استراتژی زمان واقعی عبارتند از Stonkers توسط جان گیبسون ، که در سال ۱۹۸۳ توسط Imagine Software برای ZX Spectrum منتشر شد و Nether Earth برای در سال ۱۹۸۷. در آمریکای شمالی ، قدیمی‌ترین بازی که توسط چندین منبع به عنوان استراتژی زمان واقعی طبقه بندی شده است ،The Ancient Art of War (۱۹۸٤) است که توسط Dave and Barry Murry از شرکت Evryware طراحی شده است ، و پس از آن The Ancient of War at Sea در سال ۱۹۸۷.[۲،۱۲]

در ژاپن ، اولین این نوع بازی Bokosuka Wars (۱۹۸۳) است ، یک بازی نقش‌آفرینی [٤] استراتژی اولیه (یا "نقش‌آفرینی شبیه‌سازی") ؛ این بازی حول محور بازیکنی است که ارتش را در میدان جنگ علیه نیروهای دشمن در زمان واقعی و هنگام سربازگیری در طول راه هدایت می‌کند ، که توسط Ray Barnholt از UP.com۱ به عنوان نمونه اولیه بازی استراتژی در زمان واقعی در نظر گرفته شده است. به یکی دیگر از عناوین اولیه با عناصر استراتژی در زمان واقعی ساخته Sega ، Gain Ground (۱۹۸۸) است ، یک بازی استراتژی–اکشن که شامل هدایت مجموعه ای از نیروها در سطوح مختلف پر از دشمن می‌شود. TechnoSoft's Herzog (۱۹۸۸) به عنوان پیشرو در ژانر استراتژی زمان واقعی در نظر گرفته می‌شود ، که پیشینیان Herzog Zwei بوده و از نظر ماهیتی تا حدودی مشابه است.[۱۳،۱٤،۱۵،۱٦،۱۷]

در شکل ۱ و ۲ نمایی از بازی Dune 2 و Ancient Art of War مشاهده می‌شود که از اولین بازی های در این سبک بودند. در تصویر مشاهده می‌شود که Dune 2 شامل ساختمان‌ها و ساخت است ولی در Ancient Art of War تنها تاکتیک‌ها جنگ و کنترل نیروها وجود دارد.

[4] Role-Playing Game

شکل ۱ (تصویری از Dune 2)



شکل ۲ (تصویری از Ancient Art of War)

۱-۲-۱- عناوین اصلی در سال‌های ۱۹۹۲ – ۱۹۹۸

اگرچه بازی های استراتژی زمان واقعی دارای سابقه گسترده‌ای هستند ، اما برخی عناوین بیشتر از بقیه به تعریف ژانر و انتظارات عناوین استراتژی زمان واقعی کمک کرده‌اند[۱] ، به ویژه بازی هایی که بین سال‌های ۱۹۹۲ تا ۱۹۹۸ توسط استودیو Westwood و Blizzard Entertainment منتشر شده اند.

با استفاده از Herzog Zwei، Populous، Eye of the Beholder و رابط کاربری Macintosh، Westwood's Dune II: The Building of a Dynasty (۱۹۹۲) تمام مفاهیم اصلی و مکانیک بازی‌های استراتژیک زمان واقعی مدرن را که امروزه هنوز مورد استفاده قرار می‌گیرد ، نشان می‌دهد. ، مانند استفاده از ماوس برای جابجایی واحدها و جمع آوری منابع ، و به عنوان نمونه اولیه برای بازی‌های استراتژی زمان واقعی بعدی عمل کرد. به گفته طراح و برنامه نویس اصلی ، جو بوستیک ، "مزیت آن نسبت به Herzog Zwei این است که ما مزیت ماوس و صفحه کلید را داشتیم. این امر کنترل دقیق نیروها را تا حد زیادی تسهیل کرد ، که

باعث شد بازیکن بتواند به واحدهای جداگانه دستور دهد. ماوس و کنترل مستقیم آن در ایجاد ژانر استراتژی زمان واقعی بسیار مهم بود."[۱۸،۱۹،۲۰،۲۱،۲۲]

موفقیت Dune II باعث به وجود آمدن چندین بازی شد که در نوع خود تأثیرگذار بودند.[۲،۲۱] Warcraft: Orcs & Humans (۱۹۹٤) با انتشار آن به دلیل استفاده از یک محیط فانتزی و همچنین به تصویر کشیدن انواع مختلف ساختمان‌ها (مانند مزارع) که تقریباً یک جامعه ساختگی کامل را تقلید می‌کردند ، نه فقط یک کنترل یک نیروی نظامی.Command & Conquer و همچنین Command and Conquer: Red Alert به محبوب ترین بازی‌های اولیه استراتژی زمان واقعی تبدیل شدند که نمایی از این بازی را و ساختمان های ساخته شده در آن را در شکل ۳ مشاهده می‌کنیم. این دو بازی با Warcraft II: Tides of Darkness پس از انتشار آن در اواخر سال ۱۹۹۵ رقابت کردند.

شکل ۳ (تصویری از Command and Conquer)

با پیدایش و رونق اینترنت ، بازی های استراتژی زمان واقعی جایی برای ژانر بازیکن علیه بازیکن خود در مقیاس بزرگ پیدا کردند. با بازی‌های سنتی ، مانند Civilization ، با داشتن گزینه‌ای برای چند نفره آنلاین ، سایر بازی‌های استراتژی زمان واقعی مانند Starborne: Sovereign Space ، کاملاً بر اساس مفهوم آنلاین بازیکن در مقابل بازیکن ساخته شده است.[۲۳]

در یک بازی معمولی استراتژی زمان واقعی ، صفحه نمایش به منطقه‌ای از نقشه تقسیم می‌شود که دنیای بازی و زمین ، واحدها و ساختمان‌ها را نشان می‌دهد ، و یک رابط کاربری شامل کنترل‌های فرمان و تولید و اغلب یک نمای کلی "رادار" یا "نقشه کوچک". کل نقشه معمولاً یک دیدگاه ایزومتریک از جهان یا یک دوربین آزاد از دیدگاه هوایی برای بازی‌های سه بعدی مدرن به بازیکن داده می‌شود.[۲٤] بازیکنان عمدتا روی صفحه

حرکت می‌کنند و فرمان‌ها را با ماوس صادر می‌کنند و همچنین ممکن است از میانبرهای صفحه کلید استفاده کنند.

گیم پلی به طور کلی شامل این است که بازیکن در جایی از نقشه با چند واحد یا ساختمانی که قادر به ساخت واحدها/ساختمان‌های دیگر است قرار گیرد. اغلب ، اما نه همیشه ، بازیکن باید ساختارهای خاصی بسازد تا واحدهای پیشرفته‌تری را در درخت فناوری باز کند. اغلب ، اما نه همیشه ، بازی‌های استراتژی زمان واقعی بازیکن را ملزم به ساختن ارتش (اعم از تیم‌های کوچک حداکثر دو واحد ، تا به معنای واقعی کلمه صدها واحد) می‌کند و از آن‌ها برای دفاع از خود در برابر شکل مجازی حمله موج انسانی یا از بین بردن دشمنانی که دارای پایگاه هایی با ظرفیت تولید واحد خود هستند. گاهی اوقات ، بازی‌های استراتژی زمان واقعی دارای تعدادی واحد از پیش تعیین شده برای کنترل بازیکن هستند و اجازه ایجاد واحدهای اضافی را نمی‌دهند.

جمع‌آوری منابع معمولاً تمرکز اصلی بازی‌های استراتژی زمان واقعی است ، اما عناوین دیگر این ژانر اهمیت بیشتری در نحوه استفاده از واحدها در نبرد دارند (به عنوان مثال ، Z: Steel Soldiers ، به سرزمین‌های تصرف‌شده اعتبار می‌دهد و نه منابع جمع آوری شده) ، نمونه زیاده‌روی آن‌ها بازی‌های ژانر تاکتیکی در زمان واقعی است. برخی عناوین برای تعداد سربازان همزمان سقف وضع می‌کنند ، که به یک نکته مهم در گیم پلی تبدیل می‌شود ، مثال قابل توجه StarCraft است ، در حالی که عناوین دیگر چنین محدودیتی ندارند.

۱-۳- مدیریت خرد و مدیریت کلان:

مدیریت خرد به نیاز مداوم بازیکن برای مدیریت و نگه‌داری واحدها و منابع فردی در مقیاس خوب می‌پردازد. از سوی دیگر ، مدیریت کلان به مدیریت توسعه اقتصادی و مانور استراتژیک در مقیاس وسیع اشاره می‌کند ، که به بازیکن زمان می‌دهد تا بی‌اندیشد و راه‌حل‌های احتمالی را در نظر بگیرد. مدیریت خرد شامل استفاده از تاکتیک ها می‌شود ، در حالی که مدیریت کلان مقیاس بیشتری از بازی را در تلاش برای پیش‌بینی آینده در نظر می‌گیرد.

۱-٤- مسابقات جهانی:

مسابقات جهانی استراتژی زمان واقعی از سال ۱۹۹۸ و ۲۰۰۲ برای StarCraft و Warcraft III برگزارشده است.این بازی‌ها آنقدر موفق بوده‌اند که برخی از بازیکنان در مسابقات جهانی Warcraft III بیش از ۲۰۰،۰۰۰

دلار دریافت کرده‌اند. علاوه بر این ، سالانه صدها مسابقه StarCraft II برگزار می‌شود ، چرا که این مسابقات در حال تبدیل شدن به شاخه‌ای محبوب از ورزش‌های الکترونیکی است. مسابقات قابل توجه شامل MLG، GSL و Dreamhack است. مسابقات استراتژی زمان واقعی به ویژه در کره جنوبی محبوب است.

فصل دوم

تحلیل نمونه‌های ساخته‌شده

۲-۱- مقدمه

تحلیل، بررسی و تحقیق بر روی نمونه‌های موجود و شبیه به یک پروژه به ساخت و پیاده‌سازی یک پروژه کمک شایانی می‌کند. از این رو ما در این فصل از پایان‌نامه ابتدا به تحلیل پروژه‌های ساخته شده می‌پردازیم و سپس به بررسی و مقایسه این عناوین پرداخته‌ایم. این امر باعث می‌شود از اشتباهات و موفقیت‌های آنان درس گرفته تا در پیاده‌سازی پروژه خود از آنان استفاده کنیم.

پروژه‌های تحلیل شده شش بازی معروف‌تر و پرطرفدارتر این مجموعه که بازی وارکرفت، عصر امپراطوری، قلعه، تقابل قبیله‌ها، شمال‌گرد و پادشاهان صلیبی است. این شش بازی موفقیت‌های زیاده داشته‌اند و برخی از آنان پس از سال‌های بسیار گذشته از انتشار آنان همچنان دارای طرفداران بسیاری هستند. در بررسی و مقایسه یک جدول که در آن آسان بودن رابط کاربری، وجود رقابت آنلاین، محیط و تم، رنج قیمت، پلتفرم‌ها، وجود داستان و کمپین، هوش مصنوعی قوی ملاک مقایسه قرار داده شده است.

۲-۲- تحلیل پروژه های ساخته شده توسط شرکت‌های بزرگ:

در زیر به تحلیل شش بازی بزرگ‌تر در این سبک میپردازیم.

۲-۲-۱- Warcraft III: Reign of Chaos

یک بازی رایانه‌ای فانتزی و استراتژی در زمان واقعی است که توسط Blizzard Entertainment منتشر شده است و در سال ۲۰۰۲ منتشر شد. این دومین دنباله Warcraft: Orcs & Humans ، بعد از Warcraft II: Tides of Darkness ، است. این بازی در جهان داستانی Warcraft تنظیم شده است و اولین بازی در این سری است که در سه بعد ارائه شده است. یک بسته الحاقی ، The Frozen Throne ، در ۲۰۰۳ نیز برای این بازی منتشر شد.

در این بازی ، مانند بسیاری از بازی‌های استراتژی زمان واقعی ، بازیکنان منابع را جمع آوری می‌کنند ، واحدها و قهرمانان فردی را آموزش می‌دهند و پایگاه‌هایی را برای دستیابی به اهداف مختلف (در حالت تک نفره) یا شکست دادن بازیکن دشمن ایجاد می‌کنند. چهار جناح قابل بازی را می توان انتخاب کرد: انسان ها، اورک ها (که هر دو در بازی های قبلی ظاهر شده بودند) و دو جناح جدید: الف ها و نامیرایان. کمپین تک نفره

وارکرافت شبیه به StarCraft است و به صورت پیشرونده بیان می‌شود. بازیکنان همچنین می توانند مسابقات را با رایانه یا دیگران انجام دهند – با استفاده از شبکه محلی[5] یا پلتفرم آنلاین بازی، Blizzard's Battle.net.

Warcraft III روی نقشه‌ای با اندازه‌های مختلف ، مانند دشت ها و مزارع بزرگ ، با ویژگی‌های زمین مانند رودخانه‌ها ، کوه‌ها ، دریاها یا صخره‌ها اتفاق می‌افتد. نقشه در ابتدا از دید پنهان است و فقط از طریق اکتشاف قابل مشاهده می‌شود. مناطقی که دیگر در محدوده دید واحدهای متحد یا ساختمان نیستند با مه جنگ پوشیده شده‌اند ، بدین معنا که در حالی که زمین قابل مشاهده است ، تغییراتی مانند حرکت نیروهای دشمن و ساخت و ساز ساختمان قابل مشاهده نیست. در طول یک بازی ، بازیکنان باید شهرک‌هایی برای به دست آوردن منابع ، دفاع در برابر دیگران و آموزش واحدها برای کاوش نقشه و مبارزه با دشمنان کنترل شده توسط کامپیوتر ، ایجاد کنند. سه منبع اصلی وجود دارد که در Warcraft III مدیریت می‌شوند: طلا ، چوب و مواد غذایی. دو مورد اول برای ساخت واحدها و ساختمان‌ها مورد نیاز است ، در حالی که غذا حداکثر تعداد نیروهای قابل تصرف را همزمان محدود می‌کند. علاوه بر این ، سیستم جدید "حداکثر غذا" به این معنی است که تولید واحدهای بیش از مقادیر خاص باعث کاهش مقدار طلایی می شود که بازیکنان را مجبور می‌کند تا برای جلوگیری از مجازات‌ها روی بازی با تعداد محدودی از واحدها تمرکز کنند.

بازی واحدها و ساختمان‌ها و همچنین محیط را از منظر کلاسیک از بالا به پایین با زاویه ای جزئی که فقط می توان بزرگنمایی کرد و کمی چرخاند ، نمایش می‌دهد که در شکل٤ مشاهده می‌کنیم. این بازی دارای یک رابط ثابت در پایین صفحه است که یک مینی نقشه ، اطلاعات مربوط به واحد یا گروه واحدهای انتخاب شده در حال حاضر و اقدامات احتمالی برای این واحد یا ساختمان را نشان می دهد. در صورت انتخاب چند واحد ، بازی به طور خودکار آنها را بر اساس نوع گروه بندی می‌کند و به همه واحدهای یک نوع دستورات ویژه ای داده می‌شود (مانند استفاده از مهارت های آنها). یک نوار کوچک بالا زمان فعلی روز و همچنین منابع موجود و سطح نگهداری فعلی را نشان می‌دهد. گوشه بالا سمت چپ پرتره ای از قهرمان (های) بازیکن را برای دسترسی سریع نمایش می‌دهد. اگر واحدهای کارگری کاری ندارند ، نمادهای آنها در گوشه پایین سمت چپ برای تخصیص آسان نمایش داده می شود.

[5] Local Area Network (Lan)

شکل ٤ (تصویری از Warcraft III)

Age of Empires II: The Age of Kings -۲-۲-۲

یک بازی ویدیویی استراتژی زمان واقعی است که توسط Ensemble Studios توسعه یافته و توسط مایکروسافت منتشر شده است. این بازی در سال ۱۹۹۹ برای مایکروسافت ویندوز و مکینتاش منتشر شد و دومین بازی از سری Age of Empires است. عصر پادشاهان در قرون وسطی تنظیم شده و شامل سیزده تمدن قابل بازی است. هدف بازیکنان جمع آوری منابع است که از آنها برای ساختن شهر ، ایجاد ارتش و شکست دشمنان خود استفاده می‌کند. پنج کمپین تاریخی وجود دارد که بازیکن را به شرایط تخصصی و پشتیبانی از داستان و همچنین سه حالت بازی تک نفره اضافی محدود می‌کند. چند نفره نیز پشتیبانی می‌شود.

با وجود استفاده از موتور بازی مشابه و کدی شبیه به نسخه قبلی خود ، توسعه The Age of Kings یک سال بیشتر از آنچه انتظار می‌رفت به طول انجامید و استودیوهای گروه را مجبور کرد در عوض Age of Empires: The Rise of Rome را در سال ۱۹۹۸ منتشر کنند. تیم طراحی بر حل مسائل مهمی در Age of Empires تمرکز کرد ، اما در انتشار اعلام کرد که برخی مشکلات همچنان باقی است.

Age of Empires II یک بازی استراتژی زمان واقعی است که بر ساختن شهرها ، جمع آوری منابع و ایجاد ارتش برای شکست مخالفان تمرکز دارد. بازیکنان با پیشبرد یکی از ۱۳ تمدن خود در چهار "عصر" ، شهرها و امپراتوری های رقیب را فتح می‌کنند: عصر تاریکی ، عصر فئودالی ، عصر قلعه (نشان دهنده قرون وسطی بالا) و عصر شاهنشاهی (یادآور رنسانس) - یک بازه زمانی ۱۰۰۰ ساله. پیشرفت در بازی باعث میشود

قفل واحدها ، سازه‌ها و فناوری‌های جدیدی را باز شوند ، اما بازیکنان ابتدا باید ساختمان‌هایی از عصر فعلی خود بسازند و سپس مبلغی از منابع (معمولاً غذا و طلا) را بپردازند.

از واحدهای غیرنظامی که "روستاییان" نامیده می‌شوند برای جمع‌آوری منابع استفاده می‌شود. آنها یا مرد هستند یا زن – جنسیت بر توانایی‌های آنها تأثیر نمی‌گذارد. از منابع می‌توان برای آموزش واحدها ، ساخت ساختمان‌ها و فناوری‌های تحقیقاتی و سایر موارد استفاده کرد. به عنوان مثال ، بازیکنان می‌توانند در مورد زره‌های بهتر برای واحدهای پیاده نظام تحقیق کنند. این بازی دارای چهار نوع منبع است: غذا ، چوب ، طلا و سنگ. غذا از طریق شکار حیوانات ، جمع‌آوری انواع توت ها ، برداشت دام ، کشاورزی و ماهیگیری ، هم از ساحل و هم از قایق‌ها به دست می آید که مزارع و خزانه آن را در شکل۵ مشاهده می‌کنیم. چوب با خرد کردن درختان جمع می‌شود. طلا از معادن طلا ، تجارت یا جمع‌آوری پول در صومعه به دست می‌آید. سنگ از معادن سنگ جمع‌آوری می‌شود. روستاییان برای ذخیره منابع جمع‌آوری شده به ایست بازرسی ، معمولاً ساختمان‌های انبار (مرکز شهر ، اردوگاه معدن ، آسیاب و حیاط چوب) احتیاج دارند.

عصر پادشاهان از چند نفره در اینترنت یا از طریق شبکه محلی پشتیبانی می‌کند. حداکثر هشت بازیکن می‌توانند در یک بازی شرکت کنند ، در حالی که همه حالت‌های بازی تک نفره در دسترس است. MSN Gaming Zone از بازی تا زمان بسته شدن سرویس در ۱۹ ژوئن ۲۰۰۶ پشتیبانی می‌کرد. پس از آن ، سرویس‌های مختلف بازی چند نفره مانند GameRanger از آن پشتیبانی می‌کردند. از آوریل ۲۰۱۳ ، Steam از چندنفره درون بازی که نیاز به اتصال به اینترنت دارد.

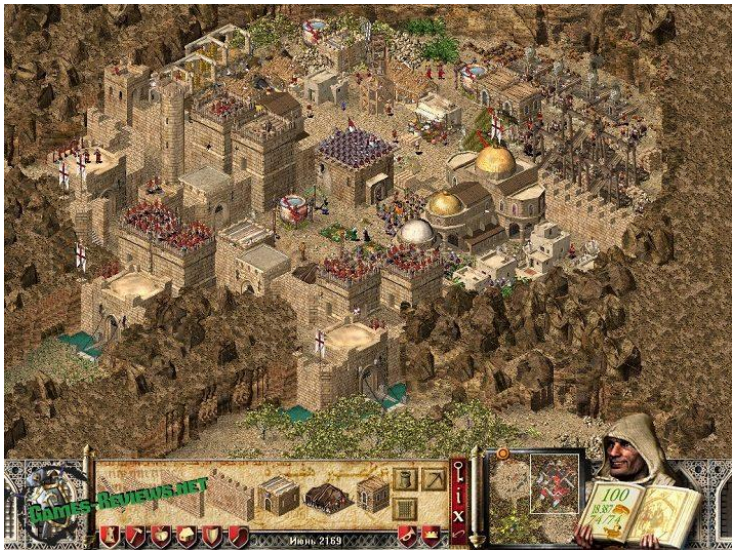
شکل ۵ (تصویری از مزارع Age of Empires II: The Age of Kings)

Stronghold: Crusader –۳–۲–۲

جانشین بازی ویدئویی استراتژیک ۲۰۰۱ Firefly Studios Stronghold است. بازی Crusader شباهت–های زیادی با Stronghold اصلی دارد ، اما با این نسخه قبلی تفاوت دارد زیرا بازی دیگر در انگلستان بازی نمی‌شود ، بلکه در خاورمیانه در طول جنگ‌های صلیبی بازی می‌شود. یکی دیگر از ویژگی‌های برجسته که در نسخه قبلی آن یافت نشد ، حالت درگیری در تک نفره است که به جای مبارزات خطی امکان نبردهای سفارشی با مخالفان هوش مصنوعی را فراهم می‌کند. این بازی به عنوان Stronghold Warchest نیز منتشر شد. این نسخه مجموعه ای از Stronghold و نسخه پیشرفته Stronghold: Crusader بود که شامل شخصیت‌های اضافی و یک Crusader Trail اضافی بود.

گیم‌پلی بازی شبیه Stronghold اصلی است ، تفاوت اصلی این است که بازی در خاورمیانه تنظیم می‌شود که در شکل٦ مساجد و قلعه‌های خاورمیانه‌ای آن قابل مشاهده است. در نتیجه ، مزارع فقط می‌توانند روی چمن ساخته شوند ، که منجر به رقابت بین بازیکنان برای محدودیت زمین‌ها و منابع می شود. این بازی مخالفان جدید هوش مصنوعی (تعداد آنها بسته به نسخه بازی) و چندین واحد جدید عربی قابل خرید اضافه می‌کند. رنگ واحدهای بازیکن نیز از آبی به قرمز تغییر یافته است تا با رنگ های شوالیه تمپلار مطابقت داشته باشد. به غیر از مزارع منابع دیگری مانند سنگ آهن ، معدن (برای سنگ) و مرداب (برای نفت) وجود دارد. این منابع بر روی انبار ذخیره می‌شود و بازیکن می‌تواند فروش یا استفاده از آن‌ها را برای اهداف دفاعی انتخاب کند. دو راه برای ایجاد ارتش وجود دارد. یا سلاح بسازید و سپس کمی طلا خرج کنید تا دهقانان را به سرباز تبدیل کنید ، یا آنها را مستقیماً به سربازان با استفاده از طلا بیشتر از طریق پست مزدور تبدیل کنید.

یکی از مشخصات این بازی وجود میزان شادی نیروهای مردمی بود که در سمت چپ پایین شکل٦ آن را می‌بینیم و اگر بازیکن مردم شادی نداشته باشد نیروهای آن کمتر تولید شده و باعث می‌شود بازیکن ارتش کوچکی داشته باشد. برای شاد کردن مردم از ساختمان‌هایی مانند پارک و مسجد و کلیسا و حفر چاه و تفریح استفاده می‌شود.



شکل٦ (تصویری از Stronghold: Crusader ظاهر خاورمیانه ای آن)

Clash of Clans -٤-٢-٢

یک بازی ویدئویی استراتژیک موبایل رایگان است که توسط توسعه‌دهنده فنلاندی Supercell توسعه و منتشر شده است. این بازی برای سیستم عامل های آی او اس در ۲ آگوست ۲۰۱۲ و در Google Play برای اندروید در ۷ اکتبر ۲۰۱۳ منتشر شد. بازی در دنیایی با تم فانتزی تنظیم می‌شود که در آن بازیکن رئیس یک روستا است. کلش آف کلنز بازیکنان را مجبور می‌کند تا با استفاده از منابعی که از حمله به روستاهای بازیکنان دیگر به دست آورده‌اند ، دهکده خود را بسازند. کسب پاداش ، خرید آن‌ها با مدال یا تولید آن‌ها در روستای خود. برای حمله ، بازیکنان انواع مختلفی از نیروها را با استفاده از منابع آموزش می دهند. منابع اصلی طلا ، اکسیر و اکسیر تیره هستند. بازیکنان می‌توانند برای ایجاد قبیله ، گروه‌هایی تا پنجاه نفر به هم متصل شوند ، که می‌توانند با هم در جنگ‌های قبیله‌ای شرکت کنند ، نیرو اهدا کنند و نیرو دریافت کنند و با یکدیگر چت کنند.

کلش آف کلنز یک بازی چند نفره آنلاین است که در آن بازیکنان اجتماعاتی به نام قبیله تشکیل می‌دهند ، نیروها را آموزش می‌دهند و برای کسب منابع به بازیکنان دیگر حمله می‌کنند. چهار ارز یا منبع در بازی وجود دارد. از طلا و اکسیر می‌توان برای ساخت و ارتقاء خطوط دفاعی و تله‌هایی استفاده کرد که روستای بازیکنان را در برابر حملات بازیکنان دیگر محافظت می‌کند و برای ساخت و ارتقاء ساختمان‌ها استفاده می شود. اکسیر و اکسیر تیره نیز برای آموزش و ارتقاء نیروها و طلسم‌ها استفاده می‌شود. جواهرات ارز برتر هستند. حملات در مقیاس سه ستاره رتبه بندی می‌شوند و حداکثر زمان آنها سه دقیقه است. این بازی همچنین دارای یک کمپین

شبه تک نفره است که در آن بازیکن می‌تواند به تعدادی از روستاهای مستعمره اجنه حمله کرده و طلا ، اکسیر (و اکسیر تیره در سطوح بالاتر) بدست آورد.

برای انجام ارتقاء ، به یک سازنده رایگان نیاز است. بازی با دو سازنده شروع می‌شود ، اما بازیکن می‌تواند با خرید و باز کردن قفل OTTO Hut در Builder Base 9 تا پنج سازنده از طریق خرید آنها و حتی یک ششم داشته باشد. در شکل۷ نمایی از بازی را که بازیکن در آن به سطح ۱۲۳ رسیده است و دارای قلعه‌ای با سطح ۸ است را مشاهده می‌کنیم.

شکل ۷ (تصویری از ظاهر فانتزی Clash Of Clans و رابط کاربری ساده آن)

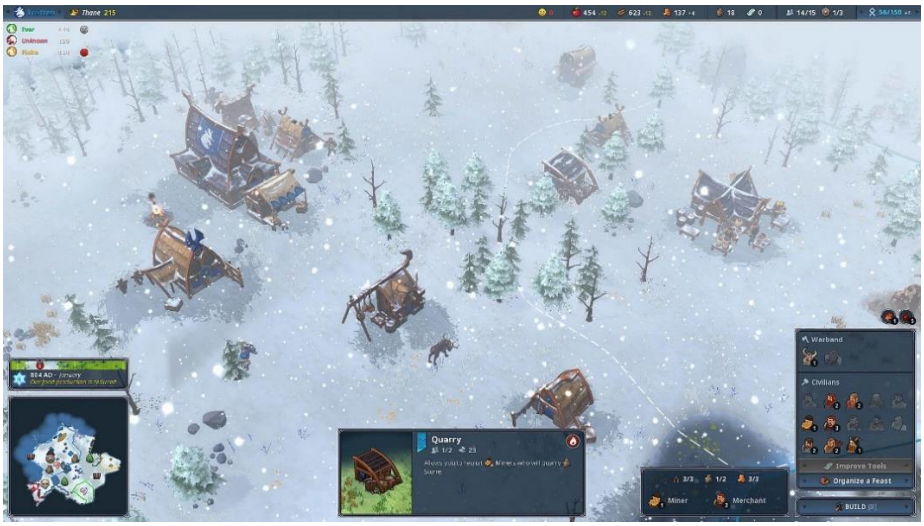
Northgard -۵-۲-۲

با الهام از اساطیر اسکاندیناوی یک بازی استراتژیک زمان واقعی است که در آن یک قبیله وایکینگ‌ها سعی می‌کنند کنترل یک قاره وحشی را به دست بگیرند. سایت سازنده بازی را اینگونه توصیف می کند: "پس از سال‌ها کاوش های خستگی ناپذیر، وایکینگ‌های شجاع سرزمین جدیدی پر از رمز و راز، خطر و ثروت را کشف کردند: Northgard. جسورترین مردم شمالی برای کشف و تسخیر این سواحل جدید کشتی را به راه انداخته‌اند و برای آنها شهرت به ارمغان می‌آورند. از طریق تسخیر، تجارت یا فداکاری به خدایان، تاریخ را قبیله کنید و بنویسید. یعنی اگر آنها بتوانند از گرگ‌های وحشتناک و جنگجویان مرده‌ای که در زمین پرسه می‌زنند جان سالم به در ببرند، با غول ها دوست شوند یا آن‌ها را شکست دهند و در سخت‌ترین زمستان‌هایی که تا به حال در شمال شاهد بوده‌اند زنده بمانند."

Northgard اولین بار در مارس ۲.۱۸ برای رایانه های شخصی منتشر شد. نسخه های کنسول پس از آن در سپتامبر ۲.۱۹ بر روی ایکس باکس وان، در سپتامبر ۲.۱۹ برای نینتندو سوییچ و در اکتبر ۲۰۱۹ برای پلی استیشن ٤ عرضه شد. نسخه موبایل نیز در آوریل ۲۰۲۱ برای دستگاه های iOS و Android منتشر می شود.

در این بازی چندین قبیله وایکینگ با کشتی به یک منطقه وارد شده و شروع به ساخت ساختمان‌های خود می‌کنند، نیروهای آن تنوع زیادی دارند و بیشتر آنان از نوع غیر سرباز هستند مانند: شکارچی ، کشاورز ، شفادهنده ، معدن‌چی ، دریانورد و برده می‌باشد. در این بازی تمرکز بیشتر بر روی ساخت یک جامعه که قسط زنده ماندن در زمستان‌های طاقت‌فرسای بازی را دارد می‌باشد و نیروهای کمتری به عنوان سرباز وجود دارد.

در این بازی همچنان که در حالت آنلاین بازیکنان دیگر وجود دارند بازیکنان هوش مصنوعی نیز وجود دارد. بازیکنان می‌توانند از بین قبیله‌های مختلف زیاد آن جناح خود را انتخاب کنند که هر کدام از آن‌ها دارای قابلیت-های خود می‌باشند برای مثال بعضی از آنان دریانوردان بهتری می‌باشند در حالی که بعضی دیگر بر روی کشاورزی تمرکز دارند. در شکل ۸ تصویری از زمستان آن که در سمت راست پایین آن فصل‌ها و زمان رسیدن آن‌ها نشان داده شده است می‌بینیم.



شکل ۸ (زمستان های Northgard)

Crusader Kings III -۶-۲-۲

یک بازی استراتژیک بزرگ و شبیه ساز سلسله است که در قرون وسطی اتفاق می افتد. بازیکنان می‌توانند تاریخ شروع ۸٦۷ یا ۱۰٦٦ را انتخاب کنند و تا سال ۱٤٥۳ بازی کنند. سلسله‌ها می‌توانند شاخه‌هایی تشکیل دهند

که سر خود را دارند و عمدتاً مستقل از سلسله والدین خود عمل می‌کنند. سران سلسله ها می‌توانند از منبع جدیدی به نام Renown برای اعمال کنترل خود بر سلسه خود استفاده کنند.

شخصیت‌ها به جای پرتره‌های دوبعدی، مدل‌های شخصیتی سه‌بعدی دارند. مانند Crusader Kings II، آنها دارای ویژگی‌هایی هستند که بر آمار و رفتار آنها تأثیر می‌گذارد. انتخاب‌هایی که برخلاف ویژگی‌های شخصیت باشد، استرس آن شخصیت را افزایش می‌دهد. سیستم ژنتیکی بازی به شخصیت‌ها اجازه می‌دهد تا برخی از ویژگی‌های خود را به فرزندان خود منتقل کنند. شخصیت‌ها می‌توانند با افزایش Dread، دست نشاندگان خود را بترسانند تا وفادار بمانند، که وقتی شخصیت اعمال بدخواهانه انجام می‌دهد، مانند اعدام یا شکنجه شخصیت‌های دیگر، افزایش می‌یابد. شخصیت‌ها می‌توانند یکی از پنج سبک زندگی را برای دنبال کردن انتخاب کنند. هر سبک زندگی دارای سه درخت مهارت است که به شخصیت ها اجازه می‌دهد مهارت‌های مربوط به آن سبک زندگی را تقویت کنند.

نقشه بازی تقریباً چهار برابر جزئیات بیشتر از نقشه بازی Crusader Kings II و کمی بزرگتر است. حکومت‌ها مستقیماً روی نقشه نشان داده می‌شوند، به این معنی که ارتش‌ها باید در اطراف نقشه حرکت کنند تا هر زیرمجموعه را در یک شهرستان محاصره کنند، که تغییری نسبت به بازگردانی‌های قبلی است. مالیات ها عمدتاً توسط پیاده نظام با کیفیت پایین متشکل از دهقانان ارائه می شود. شخصیت‌ها باید افراد مسلح را استخدام کنند تا بتوانند سربازان باکیفیت‌تری مانند تیراندازان و سواره نظام را به میدان برسانند. شخصیت‌ها می‌توانند شخصیت‌های دیگری را از دربار یا قلمرو خود با مهارت‌های رزمی قابل توجه به شوالیه تبدیل کنند که بسیار قدرتمند هستند. ۲۰ شوالیه تقریباً برابر با ۲۰۰ مالیات دهقانی است. در شکل۹ نقشه حکومت‌های این بازی را می‌بینیم که کشور ایران در سلسله سلجوخیان نیز به چشم می‌خورد.

یکی از ویژگی‌های این بازی نبودن نیاز به جنگ حتمی و قابلیت انتخاب روش‌های صلح آمیز و سیاست-مدارانه است که برای مثال با مزدوج کردن دو شخص از دو پادشاهی می‌توان پادشاهی دیگر را از رقیب به متحد تبدیل کرد. همچنین می‌توان کل بازی را به اداره پادشاهی خود پرداخت و تماشا کرد که چگونه پادشاهی‌های دیگر با یکدیگر به جنگ می‌پردازند.

شکل ۹ (نقشه بازی Crusader Kings III)

۲-۳- بررسی و مقایسه پروژه‌های مهم

حال که برخی از بازی‌های این سبک را نام برده و توضیح دادیم در زیر به بررسی و مقایسه آن‌ها با هم می‌پردازیم. در این بررسی مواردی همچون سخت یا آسان بودن رابط کاربری ، دارا بودن رقابت آنلاین ، محیط و تم بازی ، رنج قیمتی بازی ، پلتفرم‌های اجرا کننده بازی ، دارا بودن داستانی غنی و هوش مصنوعی شخصیت‌های بازی را با هم مقایسه می‌کنیم.

برای مثال دارا بودن رقابت آنلاین بخش مهمی است که باعث می‌شود بازی بعد از مدتی فراموش نشده و مانند Starcraft بعد از ۱۱ سال از عرضه آن هنوز بازیکنان خود را داشته باشند و مسابقات و جایزه‌های آنان باعث جذب طرفداران جدید نیز بشود.

جدول ۱(مقایسه بازی ها)

	Crusader Kings III	Northgard	Clash Of Clans	Stronghold	Age Of Empires	Warcraft
رابط کاربری آسان	ندارد	دارد	دارد	ندارد	ندارد	دارد
رقابت آنلاین	دارد	دارد	دارد	دارد	ندارد	دارد
محیط و تم	قرون وسطایی	حماسه اسکاندیناویایی	فانتزی	قرون وسطایی	قرون وسطایی	فانتزی بالا[6]
رنج قیمت	متوسط	پایین	رایگان(پرداخت درون برنامه ای)	پایین	بالا	بالا
پلتفرم ها	Windows , Console	Windows , Android , IOS	Android , IOS	Windows	Windows	Windows
داشتن کمپین و داستان	دارد	دارد	ندارد	ندارد	دارد	دارد
هوش مصنوعی	قوی	ضعیف	ضعیف	متوسط	قوی	متوسط

[6] High Fantasy زیرگونه ای از فانتزی است که با ماهیت حماسی محیط آن ، مضامین یا طرح آن تعریف می شود.

فصل سوم

تحلیل و پیاده سازی

۳-۱- مقدمه

در این پروژه یک بازی استراتژی زمان واقعی را مانند چندین بازی که در فصل اول ذکر کردیم پیاده‌سازی و تحلیل می‌کنیم. در این فصل ابتدا به تشریح سناریو بازی استراتژی می‌پردازیم که در آن چندین آیتم مانند منابع، ساختمان و نیروها را تعریف می‌کنیم. پس از سناریو نمودار ERD که از موجودیت، رابطه بین موجودیت‌ها و صفات موجودیت‌ها تشکیل شده‌است را توضیح می‌دهیم. پس از آن نمودار DFD که نمودار جریان داده‌ها است و در آن موجودیت‌ها، پردازش، انبار داده و جریان داده است را توضیح می‌دهیم. در ادامه نمودار State Diagram که نوعی دیاگرام برای تشریح و توضیح رفتار یک سیستم به صورت مرحله‌ای در علوم کامپیوتری و دیجیتال می‌باشد، در نمودار حالت، کلیه حالات یک ماشین در نظر گرفته شده و هر حالت با یک دایره نشان داده می‌شود. سپس شرایط آن حالت مورد بررسی قرار گرفته و بررسی می‌شود که بر اثر ورودی‌های مختلف ماشین به کدام حالت جدید می‌رود یا در حالت فعلی باقی می‌ماند. در ادامه نمودار Class یا کلاس که انواع اشیاء درون سیستم و انواع مختلف ارتباطات ساختاری آنها را نمایش می دهد. پس از آن نمودار یوزکیس دیاگرام که در ساده‌ترین تعریف، نموداری است که برهم‌کنش‌های بین کاربر یک سیستم و آن سیستم را به تصویر می‌کشد. این برهم‌کنش‌ها به منظور رسیدن به هدف خاصی در آن سیستم انجام می‌شوند. یوزکیس دیاگرام با استفاده از چند شکل مشخص شامل آدمک، دایره و جهت‌نما ترسیم می‌شود. سپس نمودار Sequence یا نمودار توالی که رفتار سیستم را مدل می‌کند. تاکید در این نمودار بر زمان و ترتیب ارسال پیام ها است. در این نمودار مجموعه ای از اشیاء با ارسال پیام با هم ارتباط برقرار می‌کنند. در ادامه نمودار Communication یا ارتباطی که تعاملات میان اشیاء یا بخش‌های دیگر را بر پایه توالی پیام‌ها، مدل‌سازی می‌کند. این نمودار، ترکیبی از اطلاعات دریافتی از نمودارهای کلاس، توالی و نمودار مورد کاربرد، را نمایش می‌دهد و نیز قابلیت توصیف ساختارهای استاتیک و رفتارهای دینامیکی یک سیستم را دارا می‌باشد. در پایان یک گانت چارت کلی از پروژه نمایش می‌دهیم که ابزاری برای مدیریت پروژه است. در واقع نمایه‌ای تصویری از کارهایی است که در طول پروژه، برنامه‌ریزی شده‌اند و بطور معمول برای برنامه‌ریزی، پیگیری و کنترل پروژه مورد استفاده قرار می‌گیرند.

در ادامه فصل قصد پیاده‌سازی پروژه را در موتور یونیتی به کمک زبان برنامه‌نویسی سی‌شارپ و Mirror که برای اتصال به شبکه و بازی با دیگر بازیکنان با این حالت است، داریم. برای این‌کار ابتدا API، Mirror را تشریح و کمی با آن آشنا می‌شویم و سپس با راه‌اندازی سیستم راه‌یابی و فرمان به نیروها پیاده سازی را شروع می‌کنیم که

این کار با استفاده از NavMesh یونیتی انجام می‌شود. پس از آن سیستم انتخاب نیروها به کمک موس را پیاده‌سازی میکنیم. بعد از آن سیستم نشانه‌گیری و آسیب‌رسانی و سلامت را می‌سازیم. سپس سیستم ساخت نیرو و ساختمان‌ها را پیاده‌سازی می‌کنیم. در ادامه سیستم برد و باخت و رابط کاربری را پیاده سازی می‌کنیم. پس از آن سیستم منابع و جمع آوری آن را ایجاد می‌کنیم. سپس کنترل دوربین و نقشه کوچک را بوجود می‌آوریم. در پایان منوهای اصلی برنامه را چیدمان کرده و کد آن را می‌نویسیم.

۳-۲- سناریو

در این پروژه چندین آیتم کلیدی وجود دارند که در زیر به شرح آن‌ها می‌پردازیم.

منابع: شامل طلا و درختان و انواع دیگری از منابع طبیعی است که بازیکن نیاز دارد آن‌ها را جمع کند تا بتواند ساختمان‌ها و نیروهای جدید بسازد.

ساختمان‌ها: سازه‌هایی که بازیکن می‌سازد تا به کمک آن بتواند ارتش ساخته و منابع جمع کند. ساختمان ها به دو نوع اصلی تقسیم می‌شوند. ۱- ساختمان‌های سربازخانه‌ای ۲- ساختمان‌های انبار منابع ۳- ساختمان اصلی. ساختمان‌های سربازخانه‌ای: ساختمان‌هایی که با کلیک کردن روی هرکدام بازیکن می‌تواند انواع مختلفی از سربازان را تعلیم داده و در قبال آن منابعی را از دست می‌دهد. انواع مختلف این نوع سربازان که در ساختمان‌های مختلف تعلیم داده می‌شوند عبارتند از: تیرکمان‌دار ، منجنیق ، سوارکار ، پیاده‌نظام سنگین ، پیاده‌نظام سبک. ساختمان‌های انبار منابع: شامل ساختمان‌هایی می‌شود که در آن برای مثال طلاها و خوب درختان ذخیره می‌شود. که بازیکن با ساختن آن‌ها می‌تواند منابع خود را افزایش دهد. ساختمان اصلی: ساختمانی است که هر بازیکن با شروع مسابقه آن را به صورت پیش‌فرض دارد که بازیکنان دیگر سعی در نابود کردن آن دارند تا بتوانند پیروز شوند.

نیروها: شامل کارگران و سربازانی می‌شود که در بالاتر ذکر شد. کارگران را می‌توان از جمله نیروهایی گفت که به جمع کردن منابع می‌پردازند مانند: معدنچی و چوب‌بر. برای بهتر متوجه شدن انواع کارگران بالا در شکل ۱۰ از هر یک از آن‌ها آورده‌ایم. سربازان که عبارتند از: تیرکمان‌دار ، منجنیق ، سوارکار ، پیاده‌نظام سنگین ، پیاده‌نظام سبک. برای بهتر متوجه شدن انواع سربازان بالا در شکل ۱۱ از هر یک از آن‌ها آورده‌ایم.

شکل۱۰) نیروهای کارگر در راست معدن‌چی و در چپ چوب‌بر


شکل ۱۱) انواع نیروهای سرباز از راست پیاده‌نظام سنگین، پیاده‌نظام سبک، تیرکماندار، سوارکار، منجنیق





رابط کاربری: عامل اصلی تعامل با بازیکن است که بازیکن به کمک آن ساختمان‌ها و نیروهای خود را اداره می‌کند. عوامل اصلی رابط کاربری این پروژه به صورت زیر است. صفحه اصلی بازی: شامل دکمه‌های میزبان که برای میزبان شدن بازیکن و تشکیل یک مسابقه می‌باشد ، متصل شدن که برای وصل شدن به یک مسابقه می‌باشد ، خروج که برای خروج از مسابقه است. سالن انتظار: که شامل نمایشی از تمام بازیکنان پیوسته به مسابقه انواع نقشه ها و دکمه شروع مسابقه برای میزبان می‌باشد. صفحه بازی: که شامل نقشه بازی همراه با منابع و بازیکنان و ساختمان اصلی آنان می‌باشد که در آن بازیکنان به مسابقه با هم می‌پردازند.

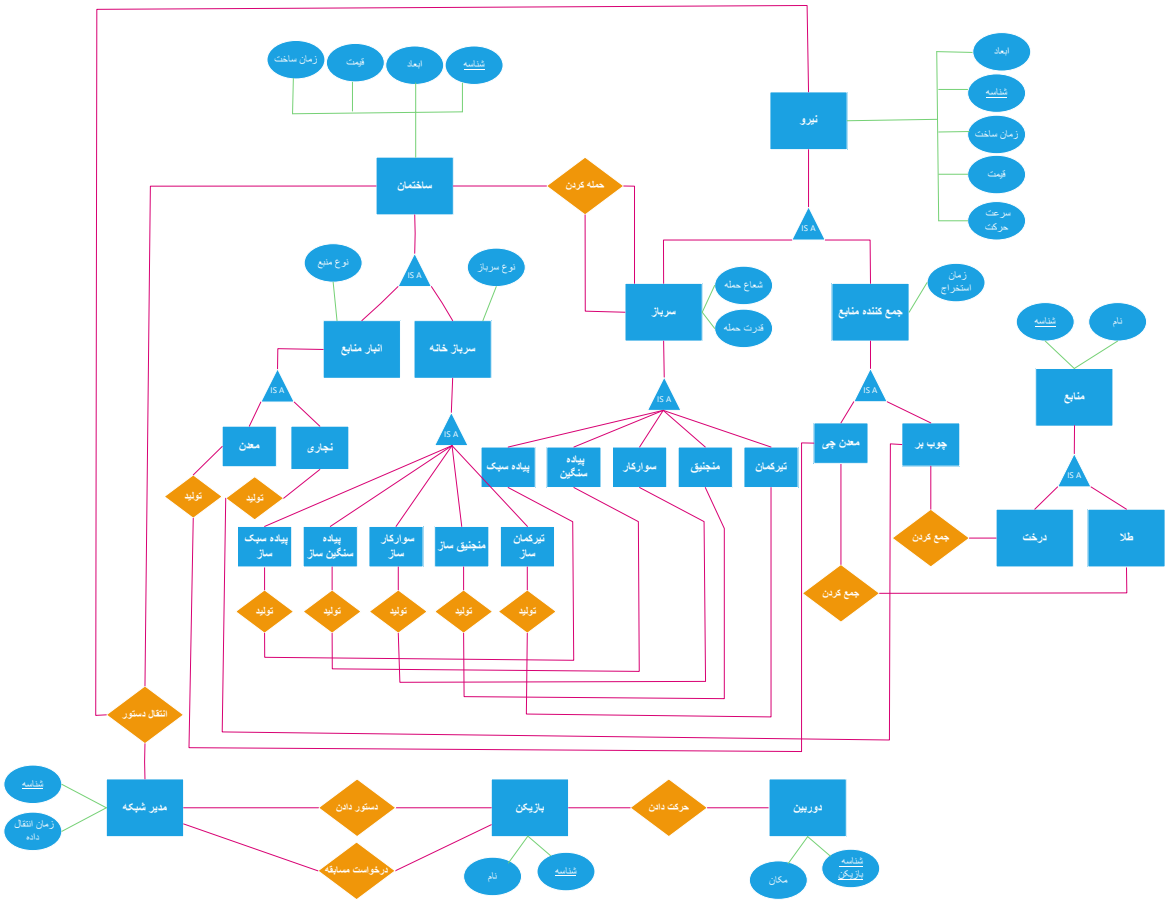
دوربین: شی‌ای در صفحه بازی است که بازیکن با آن کل نقشه بازی را مشاهده می‌کند و می‌تواند آن را حرکت داده تا از مکانی به مکان دیگر برود.

مدیر شبکه: شی‌ای که دستورات بازیکنان را از کامپیوتر آنان به سرور منتقل می‌کند و سپس به دیگر بازیکنان می‌رساند. از جمله کارهای دیگر مدیر شبکه در این پروژه اطمینان حاصل کردن از تقلب نکردن بازیکنان و ساخت و پیدا کردن مسابقه نیز است.

۳-۳- نمودار ERD

در نمودار ERD این پروژه که شامل موجودیت‌هایی همچون ساختمان که خود زیر موجودیت‌های سربازخانه و انبار منابع و ساختمان اصلی را شامل می‌شود ، نیروها که شامل سربازها و کارگرهای جمع کنندی

منابع است ، منابع ، دوربین ، بازیکن و مدیر شبکه است. بازیکنان می‌توانند مستقیم دوربین را حرکت دهند و می‌توانند به کمک ارتباط با مدیر شبکه به نیروها و ساختمان‌های خود دستوراتی را انتقال دهند ولی برای حرکت دوربین نیازی به مدیر شبکه نمی‌باشد. سربازخانه‌ها که توسط بازیکن ساخته می‌شوند انواع سربازان مختص به خود را در طول زمانی مشخص آموزش می‌دهند. انبار منابع که توسط بازیکن ساخته می‌شوند انواع کارگران را تولید می‌کنند. سربازان می‌توانند به سربازان رقیب یا به ساختمان‌های آنان حمله کرده و آنان را نابود کنند. کارگران جمع کننده منابع پس از آموزش توسط انبار منابع به جمع کردن منابع و انتقال آنان به انبار منابع میپردازند. حال در زیر نمودار ERD این پروژه را مشاهده می‌کنیم.

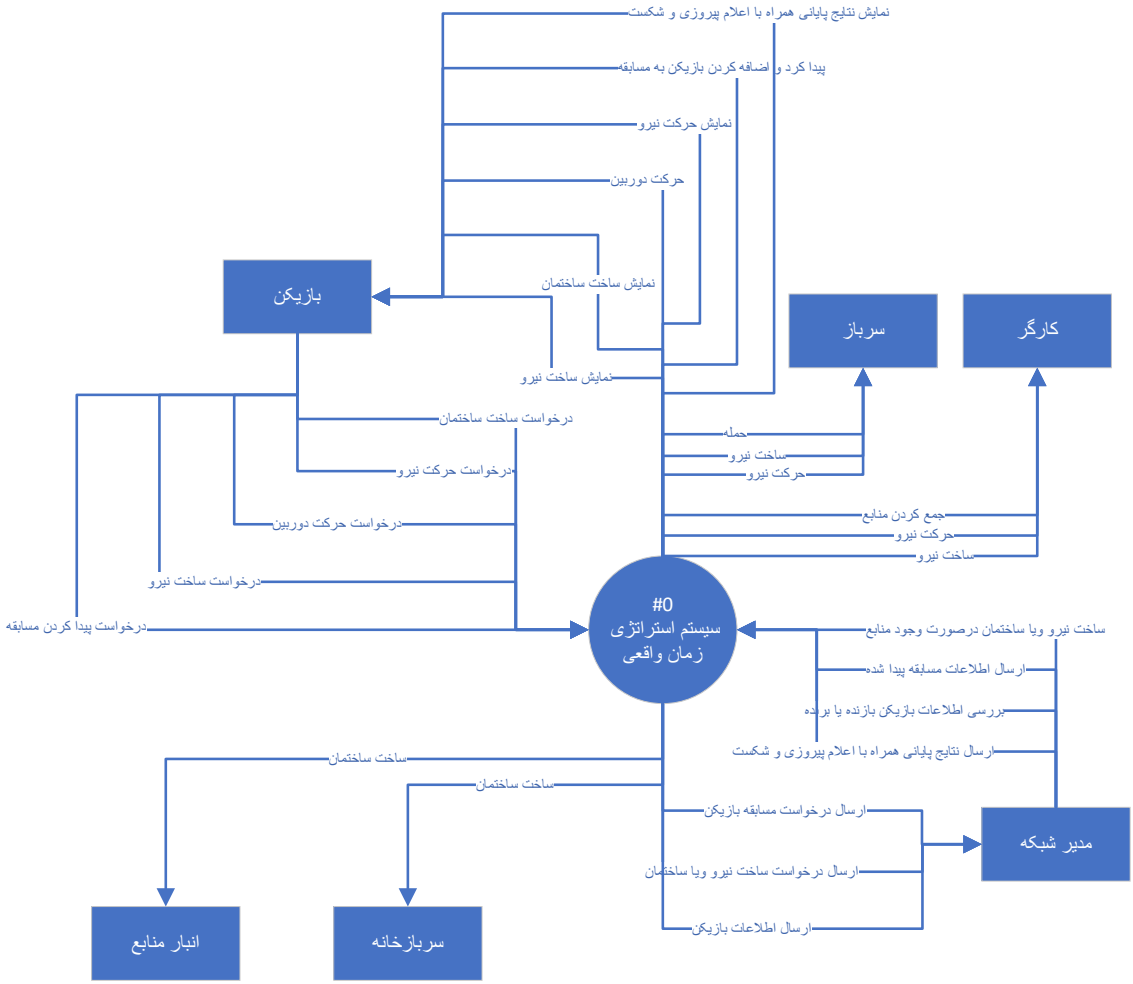


نمودار ۱ (نمودار ERD)

۳-٤- نمودار DFD

در ابتدا نمودار DFD سطح صفر در سیستم استراتژی زمان واقعی را مشاهده می‌کنیم که شامل عملیات‌های مختلفی همچون ساخت سربازان، حرکت سربازان، ساخت ساختمان‌ها، پیدا کردن و اضافه شدن به یک مسابقه

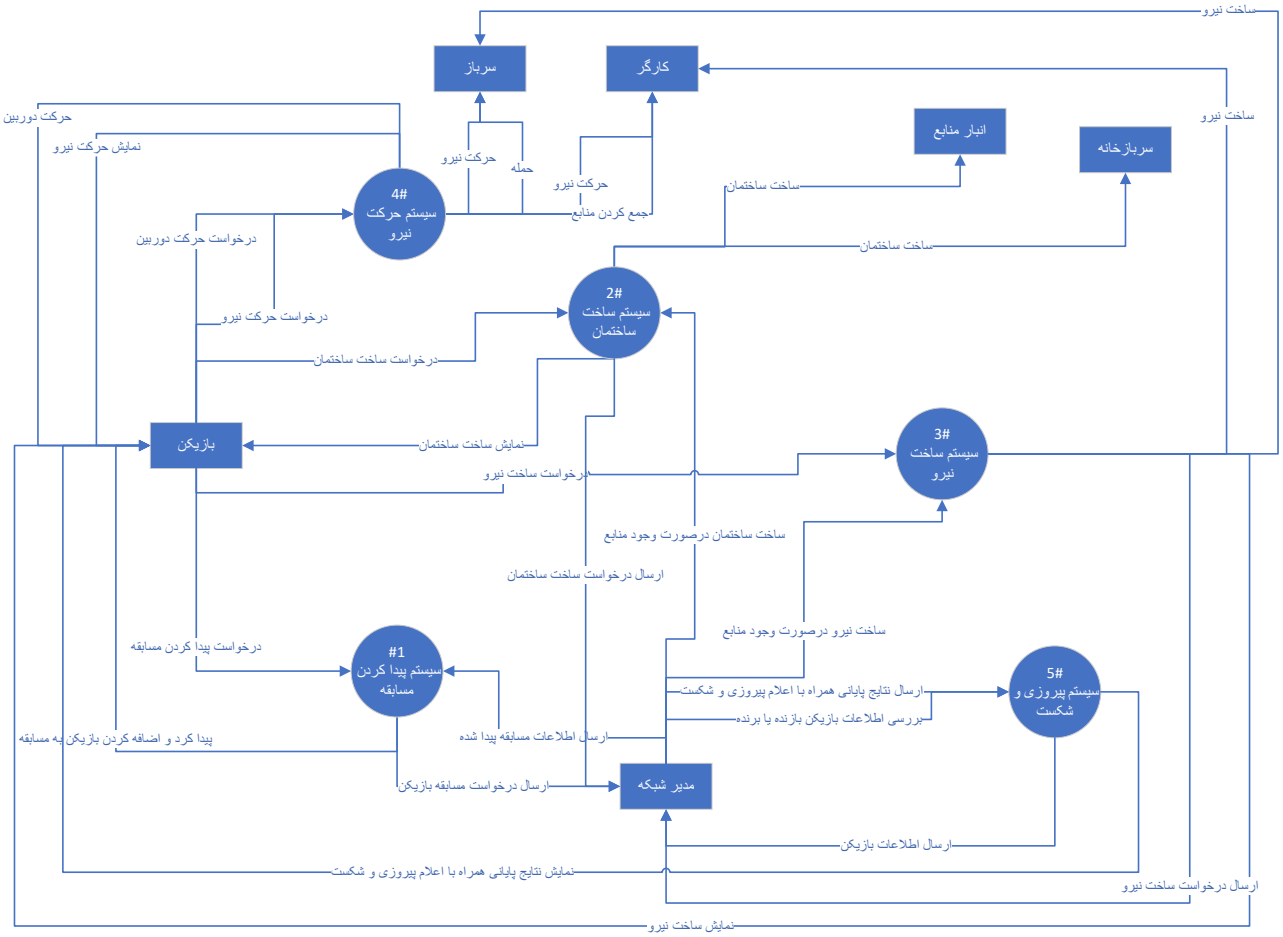
و عملیات‌های بعد از برد یا باخت می‌باشد. بازیکنان درخواست‌های خود را به سیستم ارسال می‌کند و سیستم آن درخواست‌هایی که نیاز به تایید مدیر شبکه دارد را به آن ارسال کرده و پس از تایید سیستم درخواست‌ها را انجام داده و به بازیکنان نمایش می‌دهد. برای مثال بازیکن درخواست ساخت یک ساختمان را به سیستم ارسال می‌کند ، سیستم آن را به مدیر شبکه می‌رساند ، مدیر شبکه آن را بررسی کرده و سپس درصورت وجود منابع اجازه ساخت را به سیستم می‌دهد ، سیستم ساختمان را ساخته و بعد از آن به بازیکن نمایش می‌دهد. حال در زیر نمودار DFD سطح صفر را مشاهده می‌کنیم.



نمودار ۲ (نمودار DFD سطح صفر)

حال که نمودار DFD سطح صفر را مشاهده کردیم وارد نمودار سطح یک شده و عملیات های داخل سیستم اصلی را جدا کرده و از نزدیک تر مشاهده می‌کنیم. اولین سیستم، سیستم پیدا کردن مسابقه است که در آن بازیکن درخواست پیدا کردن مسابقه را می‌دهد که سیستم آن را به مدیر شبکه منتقل کرده و بعد از آن مدیر

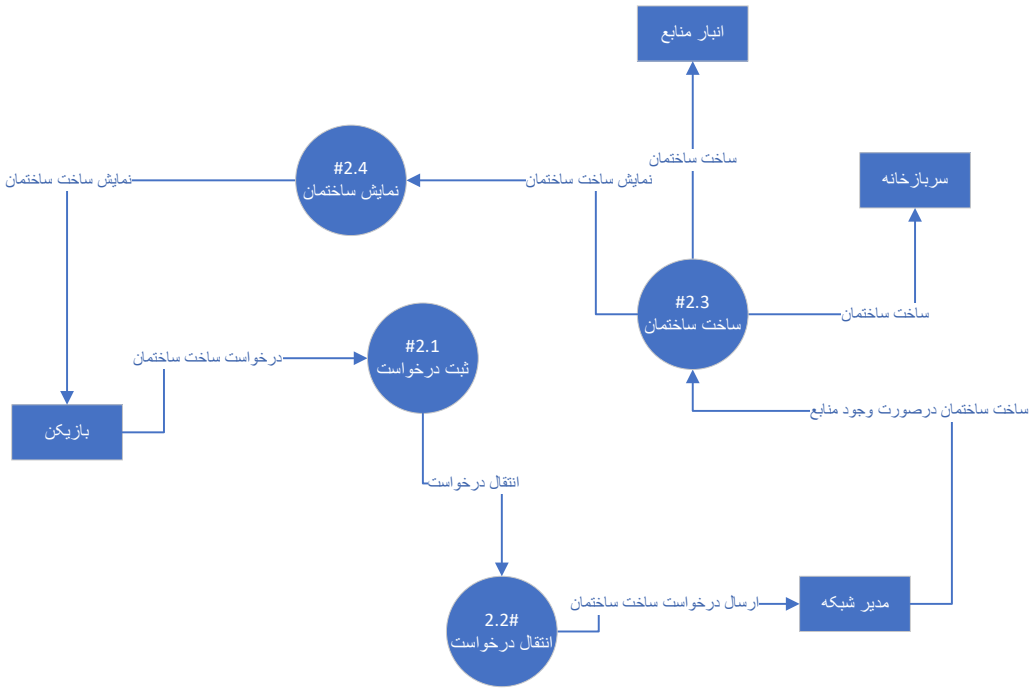
شبکه اطلاعات مسابقه پیدا شده را به سیستم فرستاده و سیستم بازیکن را به مسابقه اضافه می‌کند. دومین سیستم در نمودار سیستم ساخت ساختمان است که بازیکن در آن درخواست ساخت یک ساختمان را می‌دهد و سیستم آن را به مدیر شبکه ارسال می‌کند و مدیر شبکه در صورت وجود منابع در انبار بازیکن اجازه ساخت را به سیستم ارسال می‌کند و سیستم آن ساختمان را می‌سازد و به بازیکن نمایش می‌دهد. سومین و چهارمین سیستم که سیستم ساخت نیرو و حرکت نیرو است نیز به همین صورت کار می‌کند. پنجمین سیستم که سیستم پیروزی و شکست است به این صورت کار می‌کند که مدیر شبکه با بررسی اطلاعات بازیکنان نتایج نهایی را به سیستم ارسال می‌کند و سیستم آن را به بازیکنان نمایش می‌دهد.



نمودار ۳ (نمودار DFD سطح یک)

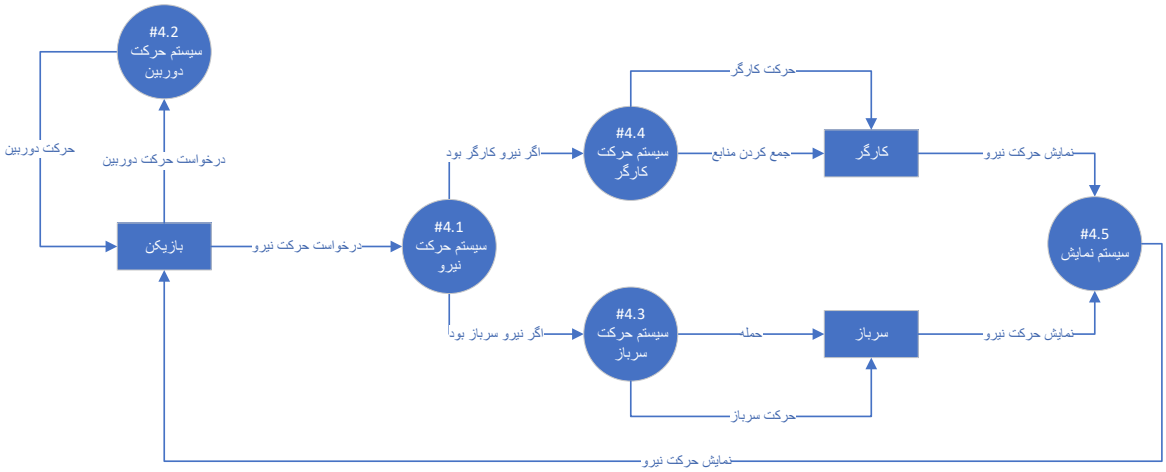
در زیر نمودار DFD سطح دو سیستم که سیستم ساخت ساختمان است را نمایش می‌دهیم که در آن بازیکن درخواست را در سیستم با کلیک روی دکمه ساختمان مورد نظر و سپس کشیدن آن به روی صفحه ثبت

کرده سپس سیستم درخواست را به مدیر شبکه انتقال داده، پس از آن که مدیر شبکه پس از بررسی وضعیت منابع بازیکن اجازه ساخت ساختمان درخواست داده شده را داد سیستم ساختمان درخواستی را ساخته و منابع مورد نیاز برای ساخت آن را از بازیکن کم کرده و سپس ساختمان ساخته شده را به بازیکن نمایش می‌دهد.



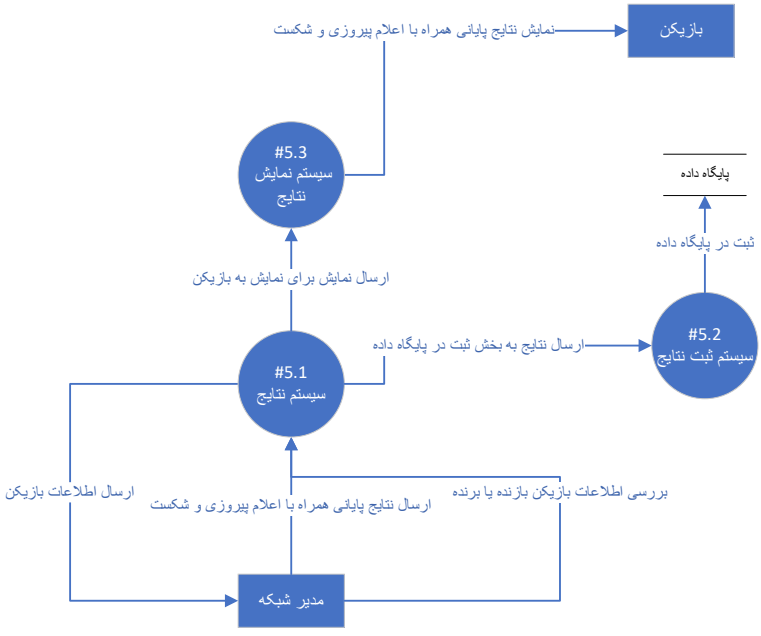
نمودار ٤ (نمودار DFD سطح دو ساخت ساختمان)

در زیر نمودار سیستم حرکت نیروها نمایش داده شده است که در آن بازیکن درخواست حرکت نیرو یا دوربین را به سیستم فرستاده و سیستم پس از بررسی مکان و نیرو دستور لازم را به نیرو داده و پس از آن به بازیکن نمایش می‌دهد. برای مثال بازیکن درخواست حمله به یکی از سربازان را می‌دهد، سیستم درخواست را دریافت کرده و بررسی می‌کند که آیا نیرو سرباز است و پس از آن مکان مشخص شده را بررسی کرده تا ببیند آیا درخواست بازیکن یک درخواست حرکت بوده یا در محل مشخص شده نیرو یا ساختمانی از رقیبان وجود دارد و درخواست در واقع درخواست حمله به آن است، پس از آن درخواست را به سرباز منتقل کرده و سپس به بازیکن حرکت سرباز را نمایش می‌دهد.



نمودار ۵ (نمودار DFD سطح دو حرکت نیرو)

در زیر نمودار DFD سطح دو سیستم که سیستم اعلام و ثبت نتایج است را مشاهده می‌کنیم که در آن مدیر شبکه پس از بررسی وضعیت بازیکنان نتایج پایانی را به سیستم اعلام می‌کند، سیستم پس از دریافت نتایج آن را در پایگاه داده سیستم و بازیکنان ثبت می‌کند تا بازیکنان در آینده بتواند نتایج قبلی خود را بازبینی کند، پس از آن نتایج را به بازیکن نمایش می‌دهد. این صفحه شامل نتایجی همچون برنده یا بازنده بودن هر بازیکن است همچنین دارای یک دکمه برای خروج به صفحه اصلی می‌باشد که بازیکنان با کلیک روی آن به صفحه اصلی بازمی‌گردند.

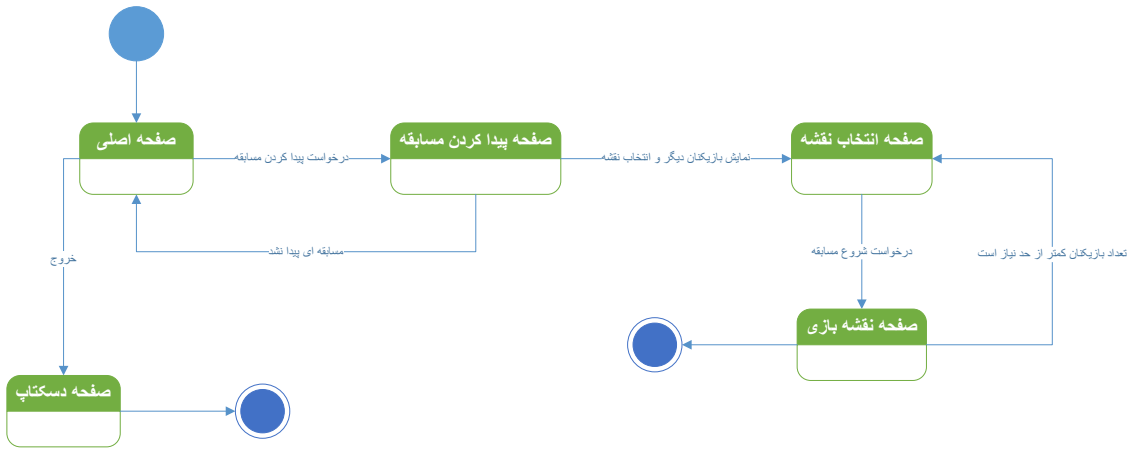


نمودار ٦ (نمودار DFD سطح دو سیستم نتایج)

۳-۵- نمودار UML State Diagram

حال به نمودار UML State Diagram می‌رسیم که رفتار سیستم را در مواجهه با یک محرک بیرونی توصیف می‌کند. در نمودار زیر که رفتار سیستم در صفحه های اصلی قبل از ورود به مسابقه بازی که بازیکن که در آن محرک بیرونی ما است در آن قصد عضو شدن به یک مسابقه و سپس شروع آن مسابقه را نشان می‌دهد.

در نمودار زیر بازیکن پس از ورود به بازی صفحه اصلی را مشاهده می‌کند که می‌تواند در آن یا از بازی خارج شده یا درخواست پیدا کردن یک مسابقه را بدهد پس از آن سیستم اگر توانست مسابقه ای را بیابد بازیکن را به صفحه انتخاب نقشه که در آن دیگر بازیکنان نیز حضور دارند می‌برد سپس پس از انتخاب نقشه توسط بازیکن مدیر مسابقه میتواند درخواست شروع مسابقه را به سیستم ارسال کند اگر تعداد بازیکنان از تعدادی خاص بیشتر باشد سیستم بازیکنان را به صفحه نقشه بازی برده و بازی شروع می‌شود. در زیر این نمودار به نمایش درآمده است.

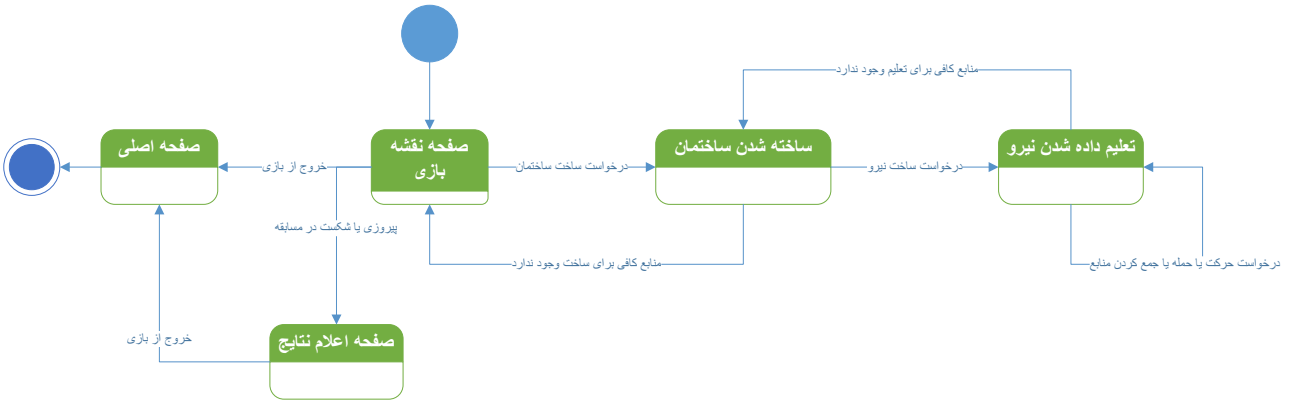


نمودار ۷ (نمودار State Diagram صفحات اصلی)

حال در زیر دومین نمودار UML State Diagram را مشاهده می‌کنیم که در آن بازیکن محرک بیرونی است و در آن عملیات های اصلی بازی پس از ورود به مسابقه نمایش داده شده است.

بازیکن پس از ورود به مسابقه ابتدا باید ساختمان‌هایی را بسازد تا به کمک آن بتواند نیرو های خود را تعلیم دهد در هر کدام از این عملیات ها اگر بازیکن منابع کافی را نداشته باشد نمی‌تواند آن عملیات را انجام دهد. پس از تعلیم نیروها بازیکن می‌تواند به کمک آن‌ها به رقیب حمله کرده و یا به جمع آوری منابع بپردازد.

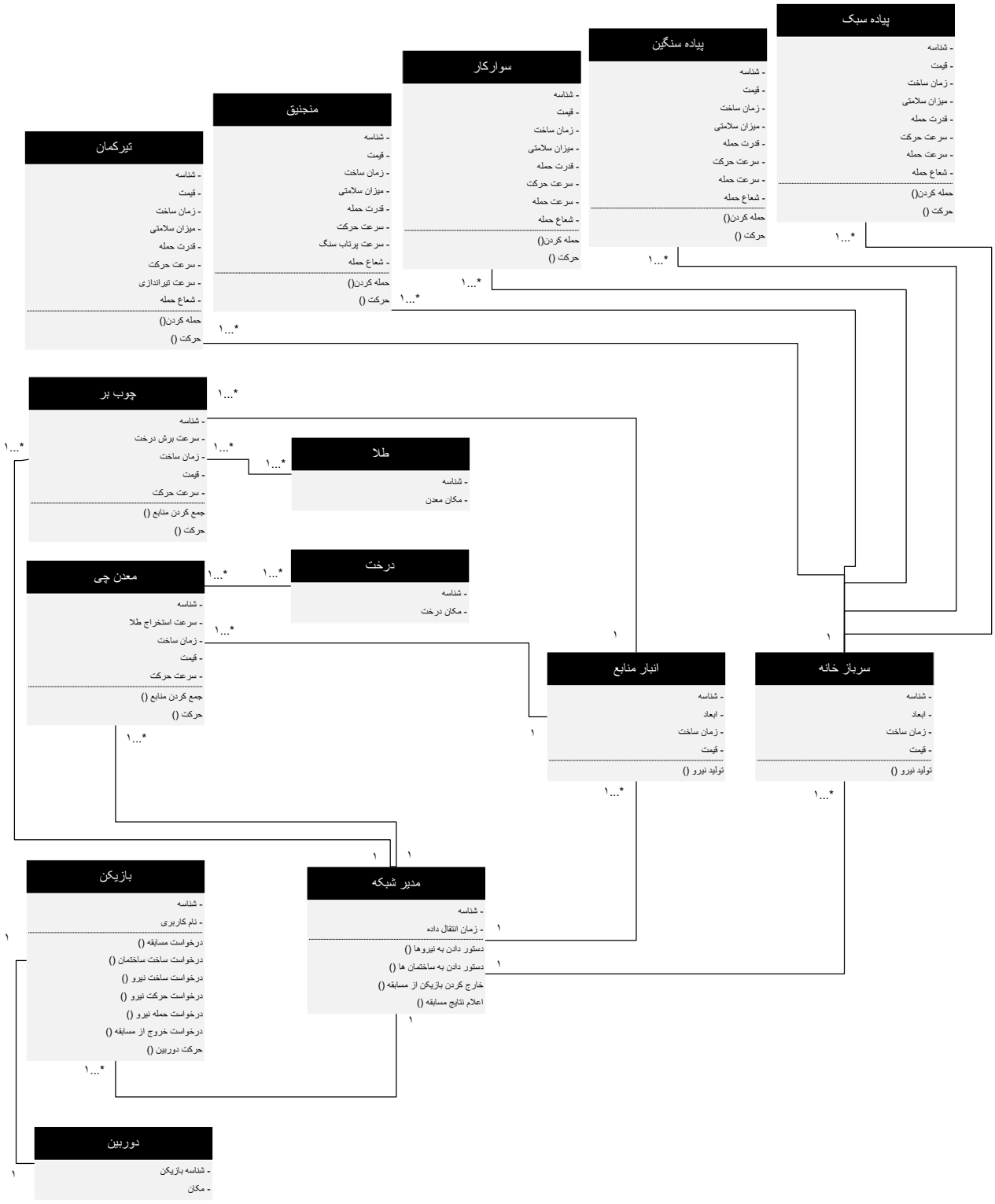
بازی تا هنگامی که تنها یکی از بازیکنان باقی بماند ادامه پیدا می‌کند و سپس صفحه اعلام نتایج مسابقه نمایش داده می‌شود و پس از آن بازیکن به صفحه اصلی بازی انتقال داده می‌شود. در زیر این نمودار را مشاهده می‌کنیم.



نمودار ۸ (نمودار State Diagram صفحه بازی)

۳-۶- نمودار Class Diagram

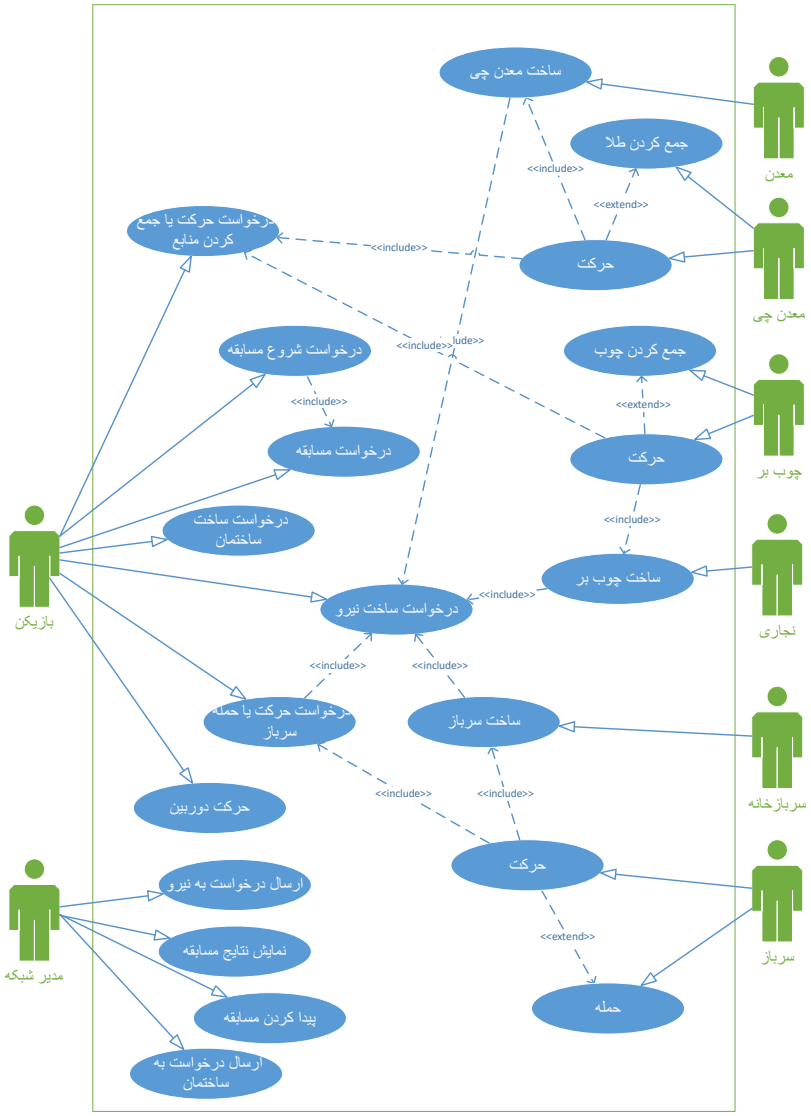
حال به نمودار Class Diagram می‌پردازیم. در این نمودار همچون نمودار ERD موجودیت های نیروها مانند سوارکار و منجنیق ، بازیکن ، ساختمان مانند سربازخانه و انبار منابع ، دوربین ، منابع مانند طلا و چوب و مدیر شبکه را داریم. کلاس بازیکن که کلاس‌هایی مانند درخواست مسابقه ، درخواست ساخت ساختمان و نیرو را دارد که آن‌ها را به مدیر شبکه منتقل می‌کند و مدیر شبکه سپس آن ها را به کلاس های مورد نظر می‌رساند ولی درخواست‌هایی مانند حرکت دوربین را دارد که به صورت مستقیم به آن منتقل می‌کند و نیازی به واسطه‌گری مدیر شبکه نمی‌باشد. کلاس‌های ساختمان‌ها با انواع کلاس نیروها در ارتباط است زیرا نیروها توسط آنان تولید می‌شوند. حال در زیر این نمودار را مشاهده می‌کنیم.



نمودار ۹ (نمودار Class Diagram)

۳-۷- نمودار Use Case

در این نمودار اکتورها همانند موجودیت‌های در نمودار ERD هستند مانند بازیکن و سرباز که به یوزکیس هایی همچون درخواست ساخت نیرو و حمله تعمیم داده شده‌اند. یوزکیس‌هایی مانند درخواست شروع مسابقه و درخواست ساخت نیرو به انجام شدن یوزکیس‌هایی همچون درخواست مسابقه و درخواست ساخت ساختمان نیاز دارند بنابراین به آن‌ها Include شده‌اند و یوزکیس‌هایی مانند جمع کردن منابع یا حمله Extend های یوزکیس‌های حرکت هستند که به این معنا است که سراحتا نیازی به آن‌ها نیست ولی می‌توانند شامل آن‌ها نیز باشند. در زیر این نمودار را مشاهده می‌کنیم.



نمودار ۱۰ (نمودار Use Case)

حال در زیر جداول Use Case را تحلیل می‌کنیم.

ابتدا یوزکیس درخواست مسابقه که در آن بازیکن اکتور است. بازیکن نیاز به داشتن نام کاربری می‌باشد تا بتواند این درخواست را به سیستم ارسال کند. پس از ارسال درخواست توسط بازیکن سیستم درخواست بازیکن را دریافت و به مدیر شبکه منتقل می‌کند، سپس مدیر شبکه درخواست را بررسی کرده و شبکه را جستجو می‌کند و پس از دریافت یک مسابقه سیستم بازیکن را به مسابقه اضافه می‌کند و صفحه لابی را به بازیکن نمایش می‌دهد.

Use Case	Description	
درخواست مسابقه	Actor	بازیکن
	Pre Condition	داشتن نام کاربری
	Main Flow	۱- بازیکن: زدن دکمه درخواست مسابقه ۲- سیستم: دریافت درخواست بازیکن و انتقال به مدیر شبکه ۳- مدیر شبکه: یافتن مسابقه و ارسال اطلاعات به سیستم ۴- سیستم: دریافت اطلاعات مسابقه و اضافه کردن بازیکن به مسابقه
	Post Condition	نمایش صفحه لابی مسابقه

جدول ۲ (جدول Use Case درخواست مسابقه)

دومین یوزکیسی که به آن میپردازیم یوزکیس درخواست شروع مسابقه توسط رییس مسابقه است که همان بازیکنی است که مسابقه را تشکیل داده است. سیستم پس از درخواست شروع رییس مسابقه درخواست را به مدیر شبکه ارسال می‌کند تا مدیر شبکه بررسی کند آیا تعداد بازیکنان کافی است یا خیر پس از تایید مکان بازیکنان را با توجه به مکان‌های پیش فرض ثبت شده در نقشه مکان‌هایی را به بازیکنان نسبت می‌دهد و پس از آن به سیستم اجازه شروع مسابقه را می‌دهد، سیستم پس از دریافت بازیکنان را به مکان‌ها مقرر در نقشه منتقل کرده و مسابقه را آغاز می‌کند.

Use Case	Description	
	Actor	رییس مسابقه
	Pre Condition	۱- تشکیل یه مسابقه

درخواست شروع مسابقه		۲- وجود بیش از یک بازیکن در لابی
	Main Flow	۱- رییس: زدن دکمه درخواست شروع مسابقه ۲- سیستم: دریافت درخواست بازیکن و انتقال به مدیر شبکه ۳- مدیر شبکه: بررسی بازیکنان مسابقه و تعیین مکان بازیکنان در نقشه و ارسال اجازه شروع به سیستم ۴- سیستم: دریافت اطلاعات مسابقه و انتقال بازیکنان به نقشه مسابقه و مکان‌های تعیین شده آنان
	Post Condition	نمایش صفحه نقشه مسابقه و شروع مسابقه

جدول ۳ (جدول Use Case درخواست شروع مسابقه)

حال به بررسی یوزکیس سوم که درخواست ساخت یک ساختمان برای مثال نجاری است می‌پردازیم. در این یوزکیس بازیکن درخواست ساخت نجاری را به صورت کلیک بر روی دکمه درخواست و کشیدن موس به روی مکانی در نقشه انجام می‌دهد، سپس سیستم درخواست را به مدیر شبکه ارسال کرده و مدیر شبکه در صورت وجود منابع در انبار بازیکن و مهیا بودن مکان برای ساخت اجازه ساخت را به سیستم می‌دهد، سپس سیستم نجاری را ساخته و از بازیکن منابع مورد نیاز را کم می‌کند.

Use Case	Description	
درخواست ساخت نجاری	Actor	بازیکن
	Pre Condition	۱- وجود بازیکن در نقشه مسابقه شروع شده ۲- وجود کافی منابع
	Main Flow	۱- بازیکن: زدن دکمه درخواست ساخت نجاری ۲- سیستم: دریافت درخواست بازیکن و انتقال به مدیر شبکه ۳- مدیر شبکه: بررسی منابع بازیکن و مکان اشاره شده توسط موس بازیکن در نقشه و ارسال اجازه ساخت ۴- سیستم: ساخت نجاری بازیکن در مکان مشخص شده و کم کردن منابع از بازیکن
	Post Condition	ساخت چوب‌بر در صورت درخواست بازیکن

جدول ٤ (جدول Use Case درخواست ساخت نجاری)

چهارمین یوزکیس پروژه درخواست حمله از طرف بازیکن به کمک چند سرباز است. در این درخواست بازیکن ابتدا باید یک یا چند سرباز ساخته شده را انتخاب کرده باشد، سپس بازیکن با کلیک بر روی محلی بر

روی نقشه درخواست را به سیستم می‌فرستد، سپس سیستم درخواست را به مدیر شبکه فرستاده تا بررسی شود که آیا مکان مشخص شده محلی قابل رفت و آمد در نقشه است و آیا ساختمان یا سربازی در مکان مشخص شده وجود دارد تا به آن حمله شود، سپس سیستم به سرباز دستور را صادر کرده و سرباز یا سربازان انتخاب شده به محل اشاره شده رفته و سپس در صورت وجود دارایی از رقیبان به آن حمله می‌کنند.

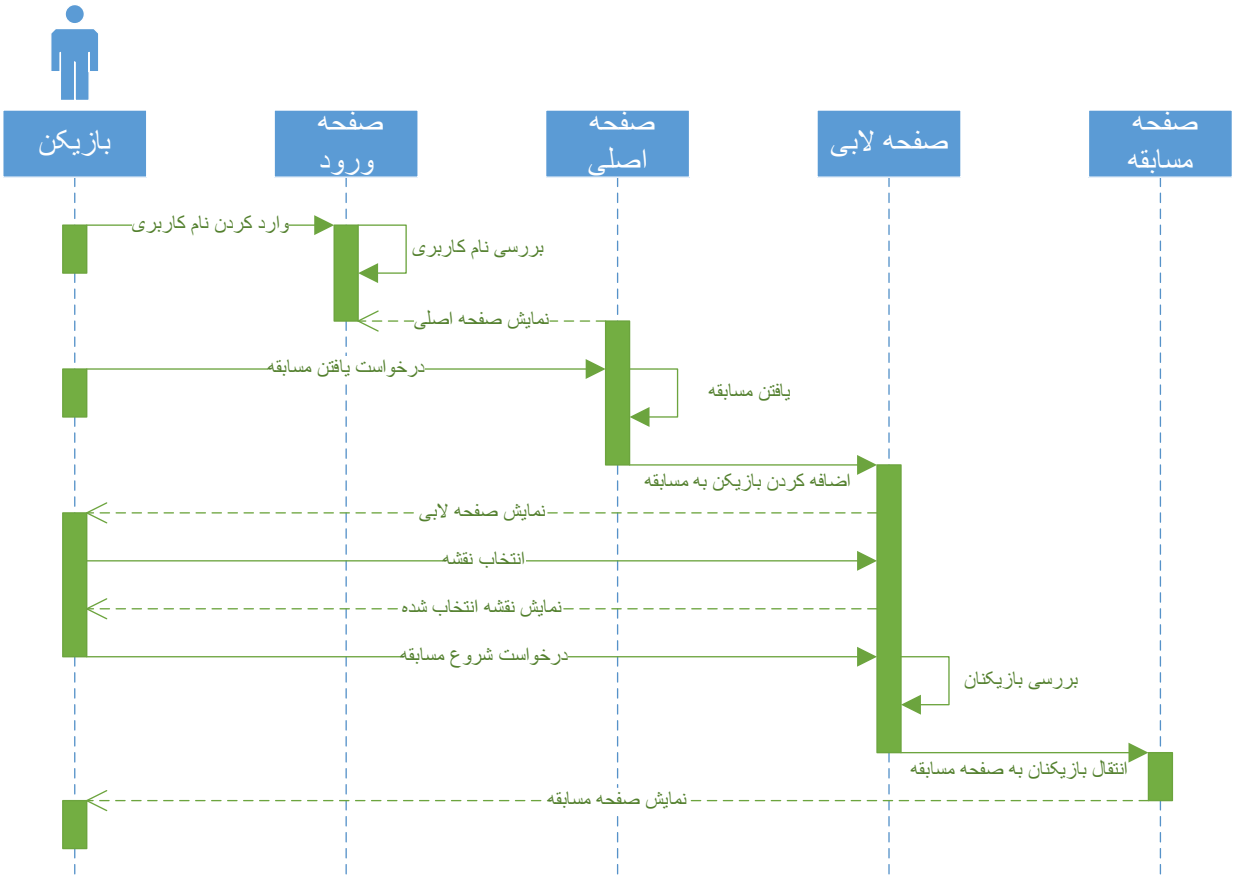
Use Case	Description	
درخواست حمله	Actor	بازیکن
	Pre Condition	۱- وجود بازیکن در نقشه مسابقه شروع شده ۲- وجود سرباز ۳- سربازان در حالت انتخاب شده باشند
	Main Flow	۱- بازیکن: درخواست حمله به سرباز ۲- سیستم: دریافت درخواست بازیکن و انتقال به مدیر شبکه ۳- مدیر شبکه: بررسی سرباز و مکان اشاره شده توسط موس بازیکن در نقشه و ارسال اجازه حمله ۴- سیستم: دستور انتقال سرباز به محل اشاره شده و حمله به سرباز یا ساختمان مشخص شده
	Post Condition	حمله سرباز به محل یا سرباز مشخص شده

جدول ۵ (جدول Use Case درخواست مسابقه)

۳-۸- نمودار Sequence Diagram

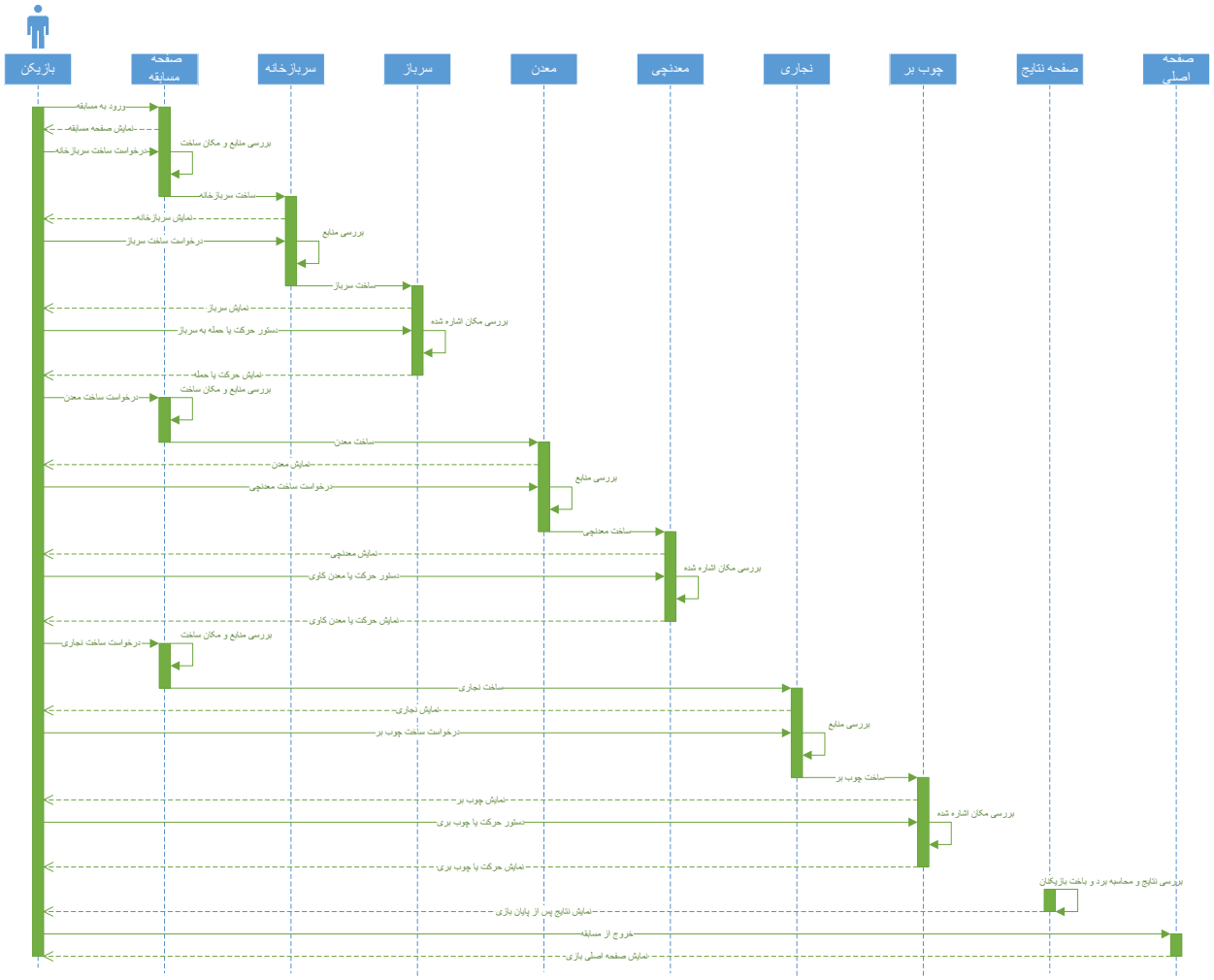
حال به نمودار ترتیب یا همان Sequence Diagram می‌رسیم. این نمودار رفتار سیستم را مدل می‌کند. تاکید در این نمودار بر زمان و ترتیب ارسال پیام‌ها است. در این نمودار مجموعه‌ای از اشیاء با ارسال پیام با هم ارتباط برقرار می‌کنند.

در نمودار اول از این مجموعه ما رفتار سیستم را در صفحه های اولیه مانند صفحه ورود و صفحه اصلی مدل کرده‌ایم. در این نمودار بازیکن ابتدا نام کاربری خود را وارد می‌کند سپس وارد صفحه اصلی شده که در آن درخواست یافتن مسابقه را می‌دهد، پس از آن در صفحه لابی نقشه مورد نظر را انتخاب کرده و دستور شروع مسابقه را می‌دهد که سیستم در صورت وجود بازیکنان کافی دستور را اجرا کرده و بازیکن را وارد صفحه مسابقه می‌کند.



نمودار ۱۱ (نمودار sequence diagram صفحات اصلی)

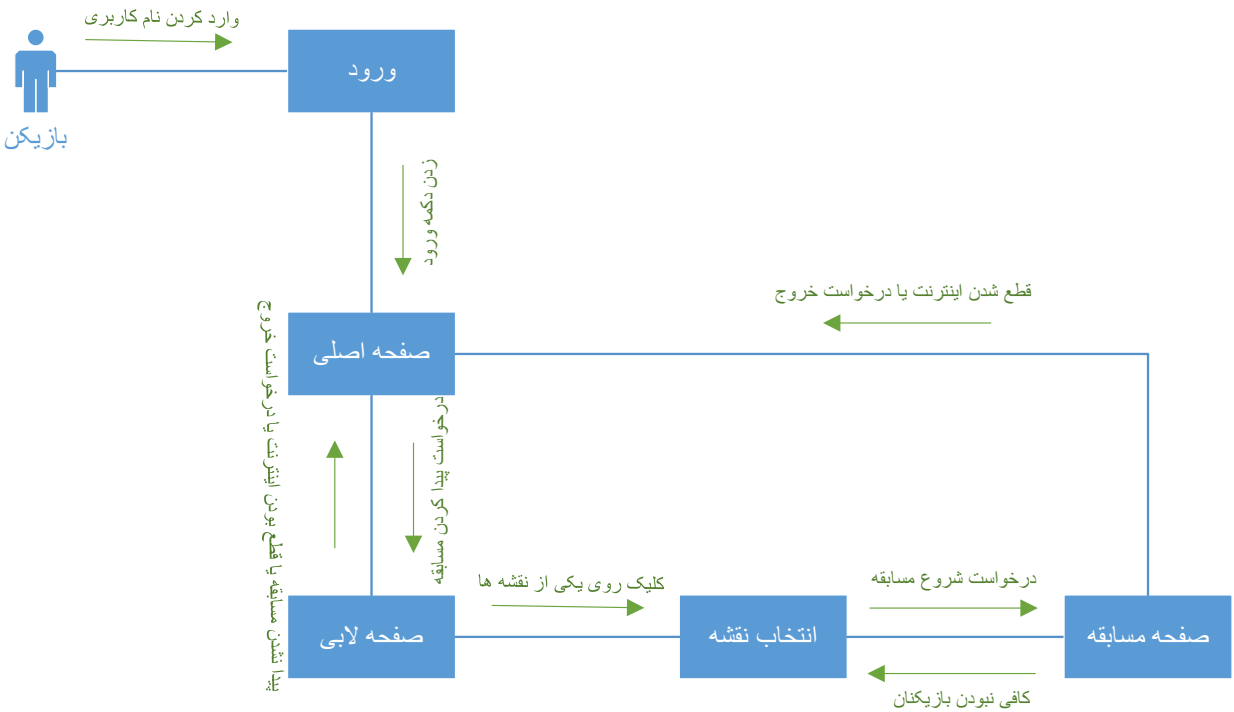
حال به دومین نمودار ترتیب اصلی این پروژه می‌پردازیم. در این نمودار رفتار سیستم در زمان مسابقه را مدل کرده‌ایم. در این نمودار بازیکن درخواست‌هایی مانند درخواست ساخت سربازخانه ، نجاری و معدن را صادر می‌کند و سیستم در صورت وجود منابع آن را می‌سازد. همچنین درخواست‌هایی مانند حمله و استخراج از معدن و چوب‌بری را که بازیکن به نیروهای خود می‌دهد نیز آمده است. یکی دیگر از درخواست‌های مهم زمان پس از اعلام نتایج است که بازیکن در آن زمان درخواست خروج از مسابقه را می‌دهد. همچنین درخواست ساخت هر نیرویی مانند چوب‌بر ، معدنچی و سربازان نیز آمده است که به این صورت است که بازیکن درخواست را به صورت کلیک بر روی ساختمانی که نیرو مورد نظر را تولید می‌کند انجام داده و سیستم در صورت وجود منابع مورد نیاز آن نیرو را تولید کرده و در همان مکان ساختمان تولید کننده می‌فرستد تا بازیکن بتواند به آن دستورات خود را صادر کند.



نمودار ۱۲ (نمودار sequence diagram صفحه بازی)

۳-۹- نمودار UML Communication diagram

حال به تحلیل نمودار UML Communication diagram میپردازیم. در اولین نموداربه تحلیل خط زندگی‌های بازیکن در صفحات ابتدایی (منوها) میپردازیم. در ابتدا بازیکن با وارد کردن نام کاربری وارد صفحه اصلی می‌شود، سپس می‌تواند درخواست پیدا کردن مسابقه‌ای را بدهد در صورت اتصال اینترنت و وجود مسابقه‌ای برای اضافه شدن سیستم بازیکن را به صفحه لابی مسابقه می‌برد که بازیکن می‌تواند در آن در صورت ادمین بودن نقشه ای را انتخاب کرده و سپس بازی را در صورت وجود تعداد کافی بازیکن شروع کند و در غیر این صورت می‌تواند از لابی خارج شده و به صفحه اصلی بازگردد. در صفحه مسابقه نیز اگر اتصال اینترنت بازیکن قطع شود یا خود بازیکن درخواست خروج از مسابقه را دهد می‌تواند از مسابقه خارج شده و به صفحه اصلی بازگردد.

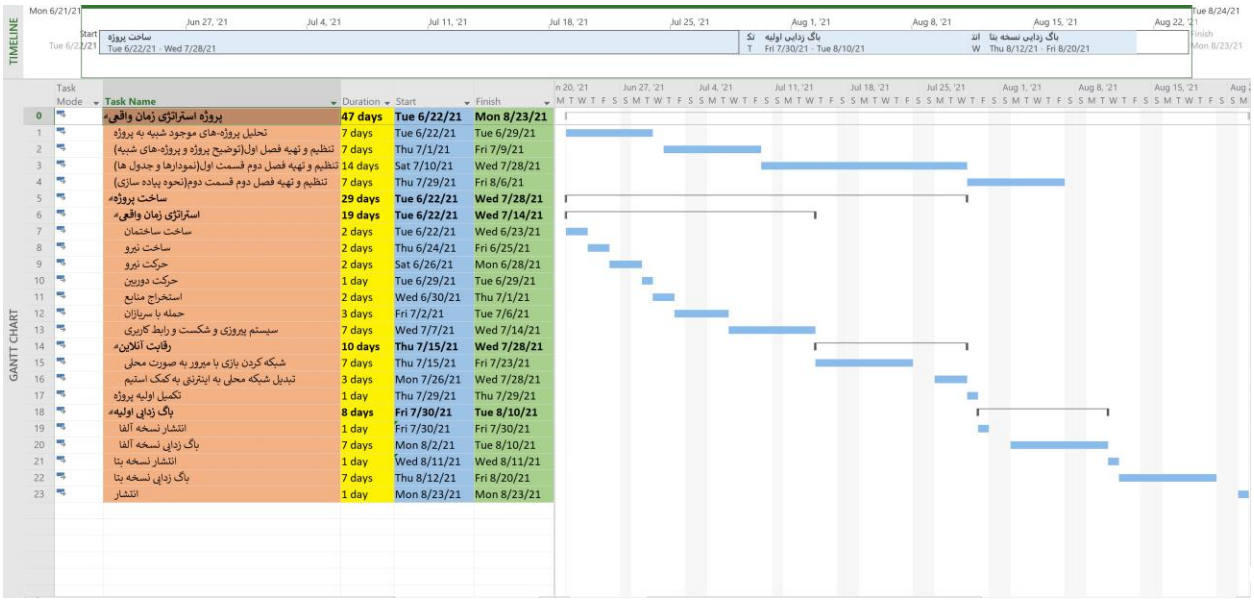


نمودار ۱۳ (نمودار Communication diagram صفحات اصلی)

۳-۱۰- نمودار گانت چارت

حال به نمودار گانت چارت این پروژه می‌رسیم که در آن خط زمانی پروژه که دارای وظایف پروژه و زمان‌های شروع و پایان آن‌ها است آمده است که تحلیل پروژه‌های موجود در ابتدا انجام شد، سپس در فصل اول پروژه‌های شبیه توضیح داده شد و پس از آن تنظیم نمودارها و جدول‌ها در قسمت اول فصل دوم قرار داده شد و پس از آن قسمت دوم این فصل که نحوه پیاده سازی است انجام خواهد شد.

ساخت خود پروژه نیز بخشی جداگانه است که به صورت موازی با فعالیت‌های بالا انجام می‌شود. در ساخت این پروژه دو بخش کلی قرار دارد که اولین آن ساخت سیستم اصلی استراتژی زمان واقعی است و بخش دیگر رقابت آنلاین است. پس از ساخت پروژه انتشار نسخه آلفا و باگ‌زدایی اولیه انجام می‌شود و پس از آن انتشار نسخه بتا و باگ‌زدایی نهایی که در نهایت به انتشار نسخه اصلی بازی می‌انجامد. در زیر این گانت چارت آمده است.



نمودار ۱۴ (نمودار Gantt chart)

۲-۱۱- پیاده‌سازی

در این پروژه همان‌طور که قبلا نیز گفته شد ما قصد ساختن یک بازی استراتژی آنلاین که بر روی پلتفورم کامپیوترها و لبتاب‌ها با سیستم عامل‌های ویندوز لینوکس و مک قابل اجرا باشد را داریم. در ساختن این پروژه از موتور بازی سازی یونیتی استفاده کرده‌ایم، که در آن کدها از نوع سی‌شارپ می‌باشند. همچنین از Mirror که یک شبکه API سطح بالا برای Unity است که از حمل و نقل‌های اطلاعاتی مختلف سطح پایین پشتیبانی می کند. همچنین از خط لوله[۷] universal render pipeline برای رندر گرفتن مدل‌ها استفاده کرده‌ایم که باعث می‌شود مدل‌ها بهتر دیده شده و نورپردازی صحنه زیباتر باشد و در عین حال توسط دستگاه‌های بیشتری قابل اجرا بوده و پردازنده کمتری مصرف کند.

ابتدا پس از درست کردن پروژه پکیج mirror را اضافه می‌کنیم. در زیر با یک مثال به توضیحات ابتدایی این پکیج می‌پردازیم. در این پکیج نوع شبکه‌ای که استفاده می‌کنیم نوع سرور–موکل[۸] است که در آن موکل(بازیکن) دستور را به سرور ارسال کرده سرور پس از چک کردن آن جواب و یا عواقب آن را به بقیه موکل‌ها انتقال می‌دهد. این نوع انتقال باعث می‌شود که دیتا کمتری استفاده شده زیرا بازیکنان تنها به سرور دیتا انتقال می‌دهند و ایمن‌تر می‌باشد و از تقلب کردن بازیکنان جلوگیری می‌کند. در شرکت‌های بزرگ سرور در

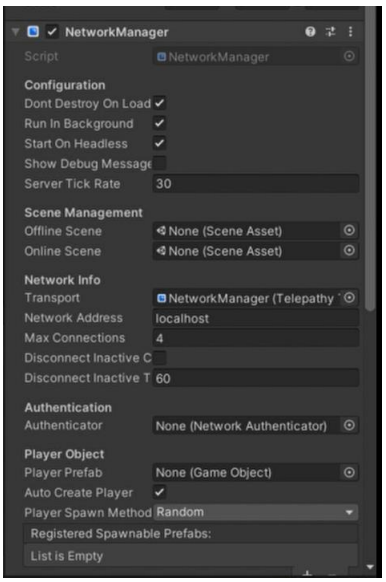
[7] Render Pipeline
[8] Client-Server

داخل شرکت قرار دارد ولی در بازی‌هایی همچون بازی ما که به صورت شخصی ساخته شده است معمولا یکی از موکلین سرور می‌شوند که در بازی ما کسی است که مسابقه را در ابتدا می‌سازد.

در عکس زیر یک بازی کوچک مثال مشاهده می‌کنیم که توسط پکیج mirror ساخته شده است، این نوع شبکه پیاده سازی شده است. همانطور که مشاهده می‌شود این بازی pong است که یک توپ بین دو بازیکن جابه‌جا می‌شود دکمه اول از سمت چپ بالا بازیکن را سرور و بازیکن می‌کند، دکمه دوم بازیکنی را به سرور متصل می‌کند و دکمه سوم کامپیوتر را فقط تبدیل به سرور می‌کند. آدرسی که در مقابل دکمه client مشاهده می‌کنیم آدرسی است که بازیکن به آن متصل می‌شود.

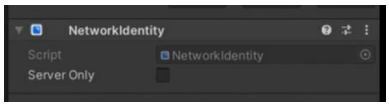
شکل ۱۲) بازی pong آنلاین با استفاده از یونیتی و mirror

در فصل قبل از شی‌ای به نام مدیر شبکه صحبت کرده و در نمودارها مشاهده کردیم این شی که در تمام صفحات بازی موجود است این شی عملیات‌هایی که با شبکه کار دارد را انجام می‌دهد. اجزای مهم آن که در شکل زیر آن را مشاهده می‌کنیم.



شکل ۱۳) اجزای تشکیل دهنده شی مدیر شبکه در mirror

در عکس بالا جز مهم آن حامل[9] است که دستگاه را به شبکه متصل کرده و دیتا را انتقال می‌دهد. جز مهم دیگر بیشترین متصلین[10] است که تعداد بازیکنانی که می‌توانند متصل شوند را نشان می‌دهد. پیش‌ساخته[11] بازیکن هم جز مهم دیگری است که زمانی که بازیکنی اضافه می‌شود سیستم یک شی با نام بازیکن جدید در صفحه می‌سازد که از این پیش ساخته استفاده می‌کند.



شکل ١٤) مولفه NetworkIdentity در Mirror

در اشیاای که با اینترنت و مدیر شبکه سروکار دارند مانند پیش ساخته بازیکن مولفه[12] بالا وجود دارد که باعث می‌شود مدیر شبکه آن‌ها را شناخته و دستوراتی که نیاز است را به آن‌ها منتقل و از آن‌ها بگیرد.

زمانی که بازیکن تلاش می‌کند به سرور متصل می‌شود در مولفه مدیر شبکه روندهایی صدا زده می‌شود که OnStartClient یکی از آنهاست که در تمام تلاش‌های بازیکن برای متصل شدن به سرور صدا زده می‌شوند درحالی که روند OnClientConnect تنها زمانی صدا زده می‌شود که بازیکن موفق به اتصال صحیح به سرور شده باشد. اتفاقات بالا در سیستم بازیکن اتفاق می‌افتد. در سرور نیز روندهایی صدا زده می‌شوند که به ترتیب OnServerConnect که زمانی صدا زده می‌شود که بازیکن موفق به متصل شدن به سرور شده است، پس از چند لحظه OnServerReady صدا زده می‌شود، پس از آن بازیکن در روند Create Player ساخته می‌شود و در آخر OnServerAddPlayer صدا زده می‌شود که بازیکن را به سرور اضافه می‌کند.

در شی بازیکن و دیگر اشیا انواعی از متغیرها وجود دارد که سرور میتواند آن‌ها را تغییر دهد این متغیرها که توسط سرور تغییر می‌دهیم و مشخص می‌کنیم به صورت SyncVar مشخص می‌شود. مثال: نام نمایش داده شده بازیکن ، رنگ بازیکن

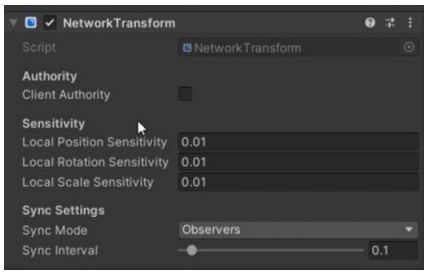
```
[SyncVar] [SerializeField] Private string displayName = “Name”;
[SncVar] [SerializeField] Private Color displayColor = Color.black;
```

[9] Transport
[10] Max Connections
[11] Prefab
[12] Component

بازیکنان می‌توانند فرمان‌هایی را در سرور صدا بزند(Command) ، همچنین سرور نیز عملیات‌هایی را می‌خواهد روی تمام دستگاه‌های بازیکنان صدا بزند.

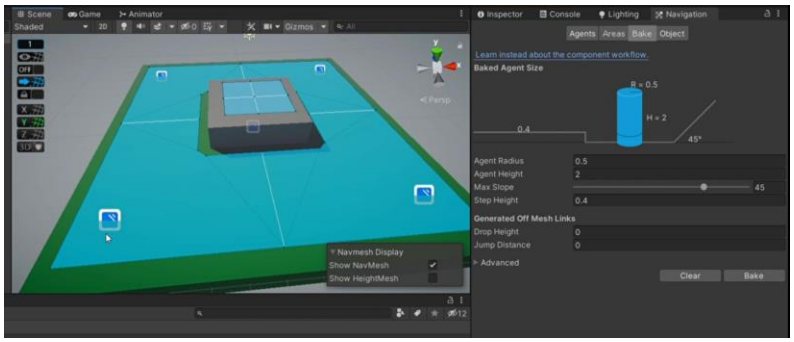
```
[Command] private void CmdSetDisplayName(string newDisplayName)
{
  SetDisplayName(newDisplayName);
}
[ClientRpc] private void RpcNewName(string newDisplayName)
{
  Debug.Log(newDisplayName);
}
```

یکی دیگر از مولفه‌ای به نام network transform است که برای بازی‌هایی که شی‌ها در آن تغییر مکان و اندازه می‌دهند همچون بازی ما که سربازان در آن حرکت می‌کنند نیاز است که باید در هر شی‌ای که تغییر مکان می‌دهد نیاز است. این مولفه را در زیر مشاهده می‌کنیم.



شکل ۱۵) مولفه network transform در Mirror

مولفه مهمی که در پروژه ما وجود دارد که برای هوش مصنوعی و راه‌یابی نیروهای داخل بازی لازم است Navigation است که در عکس زیر مشاهده می‌کنیم، مناطق آبی قابل پیمایش توسط هوش مصنوعی ما خواهد بود.



شکل ۱۶) پنل راه‌یابی در یونیتی

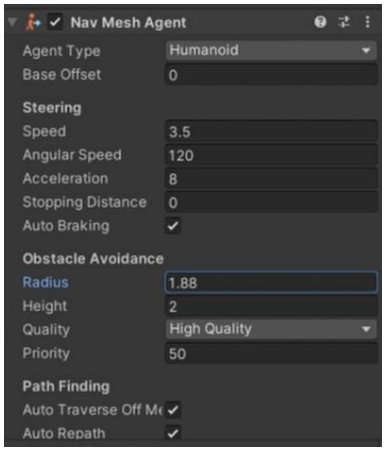
برای حرکت یک نیرو با مولفه بالا و استفاده از هوش مصنوعی Navigation توسط بازیکن به قطعه کد زیر نیاز داریم که در آن نیرو از مکان خود به نقطه position می‌رود که از SetDestination استفاده می‌کنیم.

```
[Command] private void CmdMove(Vector3 position)
{
  If (!NavMesh.SamplePosition(position, out NavMeshHit hit, 1f, NavMesh.AllAreas)) { return; }
  Agaent.SetDestination(hit.position);
}
```

برای اینکه بتوانیم با کلیک راست موس نیرو را به مکان مشخصی در صفحه بفرستیم که در بالا با رنگ آبی مکان‌های ممکن را مشخص کرده‌ایم باید از Ray استفاده کنیم که همچون شلیک کردن یک لیزر از موس به مکان داخل صفحه است و سپس پس از مشخص شدن مکان با موس کد بالا صدا زده شده و مکان به آن به عنوان position داده می‌شود، در زیر این کد را که در متد Update که در هر فریم از بازی صدا زده می‌شود مشاهده می‌کنیم.

```
[ClientCallback]
Private void Update()
{
  If (!hasAuthority) { return; }
  If (!Input.GetMouseButtonDown(1)) { return; }
  Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
  If (!Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity)) { return; }
  CmdMove(hit.point);
}
```

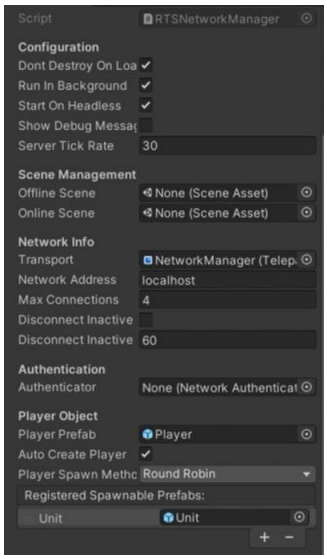
برای آنکه یک شی بتواند از Navmesh استفاده کند باید مولفه‌ای به نام Nav Mesh Agent را داشته باشد که در زیر آمده است.



شکل ۱۷) مولفه nav mesh agent در یونیتی

در این مولفه speed سرعت شی، angular speed سرعت چرخش، radius شعاع شی، height ارتفاع شی است.

حال به موضوع ساخت و تعلیم ساختمان‌ها و نیروها می‌رسیم. برای این که یک شی را بتوان در یک بازی اینترنتی که به کمک mirror ساخته شده است در طول بازی به صفحه اضافه(spawn) کرد باید آن را به قسمت Registered Spawnable Prefabs اضافه کرد که در عکس زیر می‌بینیم که یک نیرو(Unit) به آن اضافه کرده‌ایم.



شکل ۱۸) اضافه کردن یک شی به بخش Registererd Spawnable Prefab مدیر شبکه

حال یک شی ب اسم ساختمانی که می‌خواهیم نیرو از آن ساخته شود می‌سازیم و در آن یک شکل سه بعدی و یک مولفه Collider قرار می‌دهیم تا بازیکن بتواند روی آن کلیک بکند و نیروها موقع حرکت از درون آن رد نشوند.

کد UnitSpawner: حال یک کد سی‌شارپ ساخته و در آن دو متغییر شی و transform قرار می‌دهیم و در یک متد شی را که همان پیش‌ساخته نیرو ماست Instantiate می‌کنیم و پس از آن از سرور می‌خوایم تا آن را تولید کند. در عکس زیر این کد آمده است.

```
public class UnitSpawner : NetworkBehaviour, IPointerClickHandler
{
   [SerializeField] private GameObject unitPrefab = null;
   [SerializeField] private Transform unitSpawnPoint = null;
   #region Server
```

```
    [Command]
    private void CmdSpawnUnit()
    {
        GameObject unitInstance = Instantiate(
            unitPrefab,
            unitSpawnPoint.position,
            unitSpawnPoint.rotation);
        NetworkServer.Spawn(unitInstance, connectionToClient);
    }
    #endregion
```

حال متد زیر را برای آنکه هر وقت روی شی کلیک چپ موس انجام شد و بازیکن به ساختمان دسترسی داشت متد بالا را صدا بزند قرار می‌دهیم.

```
    #region Client
    public void OnPointerClick(PointerEventData eventData)
    {
        if (eventData.button != PointerEventData.InputButton.Left) { return; }
        if (!hasAuthority) { return; }
        CmdSpawnUnit();
    }
    #endregion
}
```

از آنجایی که یک بازی استراتژی دارای نیروهای بسیاری است ما می‌خواهیم هر تعداد نیرو که می‌خواهیم را انتخاب کرده و به آن‌ها دستور دهیم برای اینکار ابتدا به کد زیر که در شی نیرو قرار می‌گیرد داریم.

در این کد دو UnityEvent نیاز داریم که در موتور یونیتی به آن‌ها دو عکس که در زیر نیرو قرار می‌گیرد و زمانی که نیرو انتخاب می‌شود نمایان و زمانی که انتخاب نمی‌شود غایب می‌شود.

```
public class Unit : NetworkBehaviour
{
    [SerializeField] private UnityEvent onSelected = null;
    [SerializeField] private UnityEvent onDeselected = null;
    #region Client
    [Client]
    public void Select()
    {
        if (!hasAuthority) { return; }
        onSelected?.Invoke();
    }
    [Client]
    public void Deselect()
    {
        if (!hasAuthority) { return; }
```

```
        onDeselected?.Invoke();
    }
    #endregion
}
```

حال به یک شی که نام آن را انتخابگر نیرو می‌گذاریم نیاز داریم که در آن کد زیر به عنوان یک مولفه قرار داده شده باشد. این مولفه در ابتدا دروبین را می‌یابد، سپس در هر فریم تست می‌کند که آیا دکمه چپ موس فشار داده شده است اگر فشار داده شده بود نیروهایی که اگر قبلا انتخاب شده‌اند را رها کرده و اگر نیرویی زیر موس در صفحه بود و مطعلق به بازیکن بود آن را انتخاب می‌کند.

```
public class UnitSelectionHandler : MonoBehaviour
{
    [SerializeField] private LayerMask layerMask = new LayerMask();
    private Camera mainCamera;
    private List<Unit> selectedUnits = new List<Unit>();
    private void Start()
    {
        mainCamera = Camera.main;
    }
    private void Update()
    {
        if (Mouse.current.leftButton.wasPressedThisFrame)
        {
            foreach (Unit selectedUnit in selectedUnits)
            {
                selectedUnit.Deselect();
            }
            selectedUnits.Clear();
        }
        else if (Mouse.current.leftButton.wasReleasedThisFrame)
        {
            ClearSelectionArea();
        }
    }
    private void ClearSelectionArea()
    {
        Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());
        if (!Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)) { return; }
        if (!hit.collider.TryGetComponent<Unit>(out Unit unit)) { return; }
        if (!unit.hasAuthority) { return; }
        selectedUnits.Add(unit);
        foreach(Unit selectedUnit in selectedUnits)
        {
            selectedUnit.Select();
        }
    }
}
```

حال برای آنکه تنها به نیروی انتخاب شده دستور وارد کنیم به کدی به نام دستور دهنده به نیرو نیاز داریم که در آن مانند بالا ابتدا دوربین را پیدا می‌کنیم، سپس در هر فریم در محلی که کلیک راست کنیم اگر محل قابل رفت و آمد باشد اگر نیرویی در انتخاب ما باشد که در کد بالا ذخیره شده است آن را به محل کلیک راست شده انتقال می‌دهیم. کد ذکر شده را در زیر نمایش داده‌ایم.

```
public class UnitCommandGiver : MonoBehaviour
{
   [SerializeField] private UnitSelectionHandler unitSelectionHandler = null;
   [SerializeField] private LayerMask layerMask = new LayerMask();
   private Camera mainCamera;
   private void Start()
   {
      mainCamera = Camera.main;
   }
   private void Update()
   {
      if (!Mouse.current.rightButton.wasPressedThisFrame) { return; }
      Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());
      if (!Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)) { return; }
      TryMove(hit.point);
   }
   private void TryMove(Vector3 point)
   {
      foreach(Unit unit in unitSelectionHandler.SelectedUnits)
      {
         unit.GetUnitMovement().CmdMove(point);
      }
   }
}
```

همچنین زیرا نیازی به حرکت نیرو در کد قبلی (حرکت نیرو) نداریم و به صورت بالا آن را حرکت می‌دهیم کد به صورت زیر کاهش پیدا می‌کند.

```
public class UnitMovement : NetworkBehaviour
{
   [SerializeField] private NavMeshAgent agent = null;
   #region Server
   [Command]
   public void CmdMove(Vector3 position)
   {
      if (!NavMesh.SamplePosition(position, out NavMeshHit hit, 1f, NavMesh.AllAreas)) { return; }
      agent.SetDestination(hit.position);
   }
   #endregion
```

}

حال برای ذخیره نیروهای هر بازیکن برای پیشگیری از انتخاب نیروی یک بازیکن توسط یک بازیکن دیگر نیاز داریم تا نیروهای هر بازیکن را در یک مولفه داخل شی بازیکن به نام RTSPlayer ذخیره کنیم که به صورت زیر انجام می‌شود.

زمانی که هر بازیکن به یک مسابقه اضافه و یا از یک مسابقه خارج شوند باید نیروهای آن نیز وارد و خارج شوند که این اتفاق در AuthorityHandleUnitSpawned و AuthorityHandleUnitDespawned انجام می‌شود. زمانی که سرور شروع می‌شود نیز مانند بالا نیروها می‌توانند اضافه شوند و زمانی که سرور بسته می‌شود نیاز است تا نیروها از سرور حذف شوند که در متدهای ServerHandleUnitSpawned و ServerHandleUnitDespawned انجام می‌شود.

متدهای بالا را زمانی که سرور شروع(OnStartServer) و پایان(OnStopServer) می‌یابد و زمانی که بازیکن به بازی ملحق شده(OnStartClient) یا خروج می‌کند(OnStopClient) اضافه می‌کنیم این کد در زیر آمده است.

```
public class RTSPlayer : NetworkBehaviour
{
  [SerializeField] private List<Unit> myUnits = new List<Unit>();
  #region Server
  public override void OnStartServer()
  {
    Unit.ServerOnUnitSpawned += ServerHandleUnitSpawned;
    Unit.ServerOnUnitDespawned += ServerHandleUnitDespawned;
  }
  public override void OnStopServer()
  {
    Unit.ServerOnUnitSpawned -= ServerHandleUnitSpawned;
    Unit.ServerOnUnitDespawned -= ServerHandleUnitDespawned;
  }
  private void ServerHandleUnitSpawned(Unit unit)
  {
    if (unit.connectionToClient.connectionId != connectionToClient.connectionId) { return; }
    myUnits.Add(unit);
  }
  private void ServerHandleUnitDespawned(Unit unit)
  {
    if (unit.connectionToClient.connectionId != connectionToClient.connectionId) { return; }
    myUnits.Remove(unit);
  }
  #endregion
```

```
    #region Client
    public override void OnStartClient()
    {
        if (!isClientOnly) { return; }
        Unit.AuthorityOnUnitSpawned += AuthorityHandleUnitSpawned;
        Unit.AuthorityOnUnitDespawned += AuthorityHandleUnitDespawned;
    }
    public override void OnStopClient()
    {
        if (!isClientOnly) { return; }
        Unit.AuthorityOnUnitSpawned -= AuthorityHandleUnitSpawned;
        Unit.AuthorityOnUnitDespawned -= AuthorityHandleUnitDespawned;
    }
    private void AuthorityHandleUnitSpawned(Unit unit)
    {
        if (!hasAuthority) { return; }
        myUnits.Add(unit);
    }
    private void AuthorityHandleUnitDespawned(Unit unit)
    {
        if (!hasAuthority) { return; }
        myUnits.Remove(unit);
    }
    #endregion
}
```

تا به این لحظه انتخاب نیروها به صورت هر فرد جدا بوده است ولی از آنجایی که ما می‌خواهیم علاوه بر آن در یک بازی استراتژی نیروها را به صورت کشیدن موس و انتخاب گروهی آن‌ها انجام دهیم پس باید کد UnitSelectionHandler را تغیراتی دهیم که در زیر آمده است.

در هر فریم تست می‌کنیم که آیا کلیک چپ موس فشار داده شده است اگر فشار داده شده بود ما انتخاب کردن نیروها را آغاز می‌کنیم، سپس اگر دکمه رها شد انتخاب را پایان می‌دهیم، و در آخرین else if زمانی است که ما دکمه را فشار داده و هنوز رها نکرده‌ایم که در این زمان در هر لحظه محیط انتخاب نیرو را آپدیت می‌کنیم تا زمانی که موس رها شود.

```
    private void Update()
    {
        if (player == null)
        {
            player = NetworkClient.connection.identity.GetComponent<RTSPlayer>();
        }
        if (Mouse.current.leftButton.wasPressedThisFrame)
        {
            StartSelectionArea();
        }
        else if (Mouse.current.leftButton.wasReleasedThisFrame)
```

```
    {
        ClearSelectionArea();
    }
    else if (Mouse.current.leftButton.isPressed)
    {
        UpdateSelectionArea();
    }
}
```

حال به تحلیل StartSelectrionArea که در زمان شروع شدن فشار دادن کلیک چپ موس انجام شد می‌پردازیم. در ابتدا تمام نیروهایی که از قبل انتخاب شده‌اند را از حالت انتخاب درمی‌آوریم سپس مکان موس را خوانده و شروع به آپدیت کردن انتخاب نیروها می‌کنیم.

```
private void StartSelectionArea()
{
    foreach (Unit selectedUnit in SelectedUnits)
    {
        selectedUnit.Deselect();
    }
    SelectedUnits.Clear();
    unitSelectionArea.gameObject.SetActive(true);
    startPosition = Mouse.current.position.ReadValue();
    UpdateSelectionArea();
}
```

حال به UpdateSelectArea که در زمان نگه داشتن دکمه و حرکت موس انجام می‌شود می‌پردازیم.

یک Vector2 که در آن مکان حال موس را می‌خوانیم ذخیره می‌کنیم، در این متغییر طول و عرض موس ذخیره شده که به کمک مکان اولیه موس که در StartSelectionArea ذخیره کردیم می‌توانیم محیط این مکان را به دست آوریم و این محیط را ذخیره می‌کنیم.

```
private void UpdateSelectionArea()
{
    Vector2 mousePosition = Mouse.current.position.ReadValue();
    float areaWidth = mousePosition.x - startPosition.x;
    float areaHeight = mousePosition.y - startPosition.y;
    unitSelectionArea.sizeDelta = new Vector2(Mathf.Abs(areaWidth), Mathf.Abs(areaHeight));
    unitSelectionArea.anchoredPosition = startPosition +
        new Vector2(areaWidth / 2, areaHeight / 2);
}
```

حال به تحلیل ClearSelectionArea می‌پردازیم که در زمان رها کردن موس اتفاق می‌افتد.

در این کد ابتدا مربعی که برای کمک به دیده شدن بهتر مکان انتخابی فعال شده بود را غیرفعال می‌کنیم، سپس اگر محیط انتخاب شده صفر بود به این معنی است که بازیکن قسط انتخاب یک نیرو را داشته است که آن را انتخاب می‌کنیم و به لیست SelectedUnits اضافه می‌کنیم، اما اگر محیط بزرگتر از صفر بود تمام نیروهای در آن محیط را انتخاب می‌کنیم و به لیست SelectedUnits اضافه می‌کنیم.

```
private void ClearSelectionArea()
  {
    unitSelectionArea.gameObject.SetActive(false);
    if (unitSelectionArea.sizeDelta.magnitude == 0)
    {
      Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());
      if (!Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)) { return; }
      if (!hit.collider.TryGetComponent<Unit>(out Unit unit)) { return; }
      if (!unit.hasAuthority) { return; }
      SelectedUnits.Add(unit);
      foreach (Unit selectedUnit in SelectedUnits)
      {
        selectedUnit.Select();
      }
      return;
    }
    Vector2 min = unitSelectionArea.anchoredPosition - (unitSelectionArea.sizeDelta / 2);
    Vector2 max = unitSelectionArea.anchoredPosition + (unitSelectionArea.sizeDelta / 2);
    foreach (Unit unit in player.GetMyUnits())
    {
      Vector3 screenPosition = mainCamera.WorldToScreenPoint(unit.transform.position);
      if (screenPosition.x > min.x &&
        screenPosition.x < max.x &&
        screenPosition.y > min.y &&
        screenPosition.y < max.y)
      {
        SelectedUnits.Add(unit);
        unit.Select();
      }
    }
  }
```

در بازی‌های استراتژی معمولا زمانی که یک دسته نیرو را انتخاب کرده‌ایم میتوانیم با نگه داشتن دکمه شیفت دسته ای دیگر را به انتخاب خود اضافه کنیم، این کار را به کمک یک اضافه کردن دو خط کد زیر به کد بالا انجام میدهیم.

```
  private void StartSelectionArea()
  {
    if (!Keyboard.current.leftShiftKey.isPressed)
```

```
    {
        foreach (Unit selectedUnit in SelectedUnits)
        {
            selectedUnit.Deselect();
        }
        SelectedUnits.Clear();
    }
    unitSelectionArea.gameObject.SetActive(true);
    startPosition = Mouse.current.position.ReadValue();
    UpdateSelectionArea();
}
```

به کمک خط If(!Keyboard…) زمانی که شیفت نگه نداشته شده باشد ابتدا تمام نیروهای انتخاب شده قبلی را از لیست حذف می‌کنیم ولی زمانی که دکمه شیفت نگه داشته شده باشد این اتفاق نمی‌افتد.

```
private void ClearSelectionArea()
{
    unitSelectionArea.gameObject.SetActive(false);
    if (unitSelectionArea.sizeDelta.magnitude == 0)
    {
        Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());
        if (!Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)) { return; }
        if (!hit.collider.TryGetComponent<Unit>(out Unit unit)) { return; }
        if (!unit.hasAuthority) { return; }
        SelectedUnits.Add(unit);
        foreach (Unit selectedUnit in SelectedUnits)
        {
            selectedUnit.Select();
        }
        return;
    }
    Vector2 min = unitSelectionArea.anchoredPosition - (unitSelectionArea.sizeDelta / 2);
    Vector2 max = unitSelectionArea.anchoredPosition + (unitSelectionArea.sizeDelta / 2);

    foreach (Unit unit in player.GetMyUnits())
    {
        if (SelectedUnits.Contains(unit)) { continue; }
        Vector3 screenPosition = mainCamera.WorldToScreenPoint(unit.transform.position);
        if (screenPosition.x > min.x &&
            screenPosition.x < max.x &&
            screenPosition.y > min.y &&
            screenPosition.y < max.y)
        {
            SelectedUnits.Add(unit);
            unit.Select();
        }
    }
}
```

در خط …)if(SelecetedUnits.Contains با انجام اینکار زمانی که ما مستطیلی درست کردیم که نیروی تکراری داخلش بود آن نیرو دوباره به لیست نیروهای انتخاب شده اضافه نمی‌شود.

تا به حال در بازی ما هرگاه چند نیرو را به یک مکان بفرستیم برای رسیدن به آن مکان به هم فشار وارد کرده و پس از رسیدن به نزدیکی مکان متوقف نمی‌شدند حال برای حل این مشکل در مولفه Nav Mesh Agent متغییری به اسم Stopping Distance وجود دارد که برای استفاده از آن ابتدا به آن یک عدد می‌دهیم سپس کد زیر را به UnitMovement اضافه می‌کنیم.

با استفاده از کد زیر در زمانی که نیرو به مکانی نزدیکتر از Stoppong Distance رسید آن را متوقف می‌کنیم تا به این وسیله نیروها برای رسیدن به مرکز نقطه اشاره شده توسط بازیکن به یکدیگر فشار وارد نکرده و متوقف شوند.

```
[ServerCallback]
private void Update()
{
    if (!agent.hasPath) { return; }
    if (agent.remainingDistance > agent.stoppingDistance) { return; }
    agent.ResetPath();
}
```

حال وارد بحث نشانه گیری نیروها می‌شویم. در هر بازی استراتژی نیروها به یکدیگر و ساختمان‌ها حمله کرده و سعی در از بین بردن هم به کمک حمله از راه دور و یا نزدیک دارند. برای این کار ما به دو مولفه جدید نیاز داریم.

۱- Targetable (نشانه گرفته شده): این مولفه در شی‌هایی که نشانه گرفته می‌شوند قرار داده می‌شود مانند: ساختمان‌ها و کارگران و سربازان که به شکل زیر است.

```
public class Targetable : NetworkBehaviour
{
    [SerializeField] private Transform aimAtPoint = null;
    public Transform GetAimAtPoint()
    {
        return aimAtPoint;
    }
}
```

در کد بالا یک مولفه transform قرار دارد که محل نشانه گرفتن دیگر نیروها به این شی است که در یونیتی آن را قرار می‌دهیم. همچنین یک متد برای گرفتن این مکان در مولفه‌های دیگر با نام GetAimAtPoint است.

۲- Targeter(نشانه گیر): این مولفه در شی‌هایی که می‌خواهند نشانه گیری و حمله کنند قرار می‌گیرد مانند: سربازان ماشینالات جنگی که به شکل زیر است.

در آن یک مولفه نشانه گرفته شده قرار دارد که هدف شی است.

در متد CmdSetTarget به این صورت است که اگر شی دارای مولفه نشانه گرفته شده بود به هدف مولفه داده می‌شود.

در متد ClearTarget نیز هدف خالی گذاشته می‌شود که در زمانی مورد استفاده است که هدف نابود شده یا مقصد توسط بازیکن تغییر پیدا کرده است. این مولفه را در زیر می‌بینیم.

```
public class Targeter : NetworkBehaviour
{
   [SerializeField] private Targetable target;
   #region Server
   [Command]
   public void CmdSetTarget(GameObject targetGameObject)
   {
      if (!targetGameObject.TryGetComponent<Targetable>(out Targetable newTarget)) { return; }
      target = newTarget;
   }
   [Server]
   public void ClearTarget()
   {
      target = null;
   }
   #endregion
   #region Client
   #endregion
}
```

برای استفاده از مولفه بالا در نیروها نیاز داریم تا کد زیر را به کد UnitCommandGiver اضافه کنیم.

```
   private void Update()
   {
      if (!Mouse.current.rightButton.wasPressedThisFrame) { return; }
      Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());
```

```
    if (!Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)) { return; }
    if (hit.collider.TryGetComponent<Targetable>(out Targetable target))
    {
        if (target.hasAuthority)
        {
            TryMove(hit.point);
            return;
        }
        TryTarget(target);
        return;
    }
    TryMove(hit.point);
}
```

در کد بالا یک اگر به کد اضافه شده که در آن اگر شی‌ای که به آن اشاره شده مولفه Targetable را داشت و بازیکن ما مالک آن نبود TryTarget که به معنی سعی در نشانه گیری است انجام می‌شود که متد دیگری است که باید به کد UnitCommandGiver اضافه شود که در زیر آمده است.

```
private void TryTarget(Targetable target)
{
    foreach (Unit unit in unitSelectionHandler.SelectedUnits)
    {
        unit.GetTargeter().CmdSetTarget(target.gameObject);
    }
}
```

در این کد همانند متد TryMove که در بالاتر برای حرکت آمده بود باید به ازای هر نیروی انتخاب شده ما در آن نیرو CmdSetTarget که در بالاتر برای گرفتن مولفه Targetable آمده بود صدا زده می‌شود.

حال برای آنکه نیرو پس از نشانه‌گیری به حرکت به سمت شی نشانه‌گیری شده حرکت کند نیاز به تغییر کد UnitMovement است. باید در قسمتی که هر فریم تغییر می‌کند قسمت زیر را اضافه کنیم. ابتدا از مولفه Targeter اگر شی‌ای نشانه گیری شده باشد می‌گیریم، سپس اگر شی نشانه‌گیری وجود داشت، اگر مکان نیرو ما فاصله‌ای بیشتر از شعاع دنبال کردن باشد نیرو به سمت شی حرکت می‌کند و پس از رسیدن به شعاع دنبال کردن توقف می‌کند.

```
[ServerCallback]
private void Update()
{
    Targetable target = targeter.GetTarget();
    if (target != null)
    {
```

```
        if((target.transform.position - transform.position).sqrMagnitude > chaseRange * chaseRange)
        {
            agent.SetDestination(target.transform.position);
        }
        else if(agent.hasPath)
        {
            agent.ResetPath();
        }
        return;
    }
```

پس از نشانه گیری هدف و حرکت به سمت آن حال به قسمت شلیک به سمت هدف رسیده‌ایم برای این‌کار نیاز داریم تا یک شی برای شلیک بسازیم. پس از ساخت یک شی نیاز داریم مولفه UnitProjectile را که در زیر توضیح می‌دهیم به آن اضافه کنیم.

```
public class UnitProjectile : NetworkBehaviour
{
    [SerializeField] private Rigidbody rb = null;
    [SerializeField] private float destroyAfterSeconds = 5f;
    [SerializeField] private float launchForce = 10f;
    void Start()
    {
        rb.velocity = transform.forward * launchForce;
    }
    public override void OnStartServer()
    {
        Invoke(nameof(DestroySelf), destroyAfterSeconds);
    }
    [Server]
    private void DestroySelf()
    {
        NetworkServer.Destroy(gameObject);
    }
}
```

در کد بالا rb یک متغییر از نوع Rigidbody است که یک مولفعه فیزیکی برای تشخیص برخورد و اعمال جاذبه روی جسم است. destroyAfterSeconds یک متغیر از نوع عدد اعشاری است که زمان نابودی گلوله پس از به وجود آمدن است. launchForce یک متغیر از نوع عدد اعشاری است که سرعت پرتاب گلوله را نشان می‌دهد.

در شروع که با به وجود آمدن گلوله شروع می‌شود گلوله را به سمت روبرو با سرعت پرتاب بالا شلیک می‌کنیم.

در OnStartServer دستور DestorySelf که یک متد برای نابودی گلوله است را پس از زمان نابودی مشخص شده صدا می‌کنیم.

حال مولفه زیر را ساخته و به نیروی مهاجم اضافه می‌کنیم. در مولفه زیر targeter مولفه نشانه گیر داخل نیرو است، projectilePrefab شی پیش‌ساخته گلوله ماست، projectileSpawnPoint مکان به وجود آمدن گلوله در نیرو است، fireRange شعاع شلیک گلوله است، fireRate دوره تناوب شلیک گلوله است، rotationSpeed میزان چرخش نیرو در زمان شلیک است.

ابتدا هدف را از مولفه targeter می‌گیریم سپس اگر هدفی وجود داشت و می‌توانستیم به آن شلیک کنیم(داخل شعاع شلیک باشد) نیرو مهاجم به سمت هدف در صورت نیاز میچرخد و در دوره تناوب شلیک گلوله را شلیک می‌کند.

```
public class UnitFiring : NetworkBehaviour
{
  [SerializeField] private Targeter targeter = null;
  [SerializeField] private GameObject projectilePrefab = null;
  [SerializeField] private Transform projectileSpawnPoint = null;
  [SerializeField] private float fireRange = 5f;
  [SerializeField] private float fireRate = 1f;
  [SerializeField] private float rotationSpeed = 20f;
  private float lastFireTime;
  [ServerCallback]
  private void Update()
  {
    Targetable target = targeter.GetTarget();
    if (target == null) { return; }
    if (!CanFireAtTarget()) { return; }
    Quaternion targetRotation =
      Quaternion.LookRotation(target.transform.position - transform.position);
    transform.rotation = Quaternion.RotateTowards(
      transform.rotation, targetRotation, rotationSpeed * Time.deltaTime);
    if (Time.time > (1 / fireRate) + lastFireTime)
    {
      Quaternion projectileRotation = Quaternion.LookRotation(
        target.GetAimAtPoint().position - projectileSpawnPoint.position);
      GameObject projectileInstance = Instantiate(
        projectilePrefab, projectileSpawnPoint.position, projectileRotation);
      NetworkServer.Spawn(projectileInstance, connectionToClient);
      lastFireTime = Time.time;
    }
  }
  [Server]
  private bool CanFireAtTarget()
  {
```

```
        return (targeter.GetTarget().transform.position - transform.position).sqrMagnitude
            <= fireRange * fireRange;
    }
}
```

حال که گلوله شلیک شده و به سمت شی می‌رود و برخورد می‌کند باید کدی برای میزان سلامتی و کم کردن آن بنویسیم.

در مولفه Health که سلامتی هر شی است maxHealth بیشترین میزان سلامتی است و currentHealth سلامتی حال حاضر آن است.

در OnStartServer که زمانی که شی بی وجود می‌آید اجرا می‌شود سلامتی حال حاضر برابر با بیشترین سلامتی است.

در DealDamage که در گلوله صدا زده می‌شود اگر سلامتی صفر نبود سلامتی به میزان صدمه‌ای که گلوله وارد می‌کند کم می‌شود و اگر سلامتی برابر صفر بود ServerOnDie که زمان نابودی شی است صدا زده می‌شود. این مولفه را در زیر می‌بینیم.

```
public class Health : NetworkBehaviour
{
    [SerializeField] private int maxHealth = 100;
    [SyncVar]
    private int currentHealth;
    public event Action ServerOnDie;
    #region Server
    public override void OnStartServer()
    {
        currentHealth = maxHealth;
    }
    [Server]
    public void DealDamage(int damageAmount)
    {
        if (currentHealth == 0) { return; }
        currentHealth = Mathf.Max(currentHealth - damageAmount, 0);
        if (currentHealth != 0) { return; }
        ServerOnDie?.Invoke();
        Debug.Log("We Died");
    }
    #endregion
    #region Client
    #endregion
}
```

حال باید تغییراتی در کد UnitProjectile بدهیم تا در زمان برخورد سلامتی نیرو یا ساختمان برخورد کرده را کاهش دهد.

باید به کدی که قبل داشتیم متد OnTriggerEnter را اضافه کنیم که در زمان برخورد با اجسام دیگر صدا زده می‌شود. در این کد ابتدا چک می‌کنیم که اگر با نیرو یا ساختمانی که متعلق به بازیکن صاحب نیرو است برخورد کرده است اتفاقی نیافتد، در غیر این صورت مولفه سلامتی شی را گرفته و DealDamage که در بالا ذکر کرده‌ایم را صدا می‌زنیم و پس از آن خود گلوله را نابود می‌کند. در زیر این مولفه را می‌بینیم.

```
public class UnitProjectile : NetworkBehaviour
{
   [SerializeField] private Rigidbody rb = null;
   [SerializeField] private int damageToDeal = 20;
   [SerializeField] private float destroyAfterSeconds = 5f;
   [SerializeField] private float launchForce = 10f;
   void Start()
   {
      rb.velocity = transform.forward * launchForce;
   }
   public override void OnStartServer()
   {
      Invoke(nameof(DestroySelf), destroyAfterSeconds);
   }
   [ServerCallback]
   private void OnTriggerEnter(Collider other)
   {
      if (other.TryGetComponent<NetworkIdentity>(out NetworkIdentity networkIdentity))
      {
         if (networkIdentity.connectionToClient == connectionToClient) { return; }
      }
      if(other.TryGetComponent<Health>(out Health health))
      {
         health.DealDamage(damageToDeal);
      }
      DestroySelf();
   }
   [Server]
   private void DestroySelf()
   {
      NetworkServer.Destroy(gameObject);
   }
}
```

حال به مرحله قرار دادن یک تصویر بالای هر شی‌ای که دارای سلامتی است رسیده‌ایم که بتوانیم سلامتی نیروها و ساختمان‌های خود و رقیب را مشاهده کنیم. برای این کار به یک مولفه جدید به نام HealthDisplay داریم که در زیر مشاهده می‌کنید.

در این مولفه ابتدا یک مولفه سلامتی که برای هر کدام از نیروها و ساختمان‌ها است قرار می‌دهیم، سپس یک شی که شی پدر تصویر نمایش سلامتی ما است، پس از آن یک تصویر که سلامتی ما را نمایش می‌دهد.

در زمان قرار گرفتن موس بر روی نیرو یا ساختمان شی پدر را فعال می‌کنیم تا تصویر نمایش داده شود و در زمان خارج کردن موس شی را غیر فعال می‌کنیم. در متد HandleHealthUpdated نیز عرض عکس را به مقدار سلامتی حال حاضر تقسیم بر کل میزان سلامتی می‌کنیم و نمایش می‌دهیم.

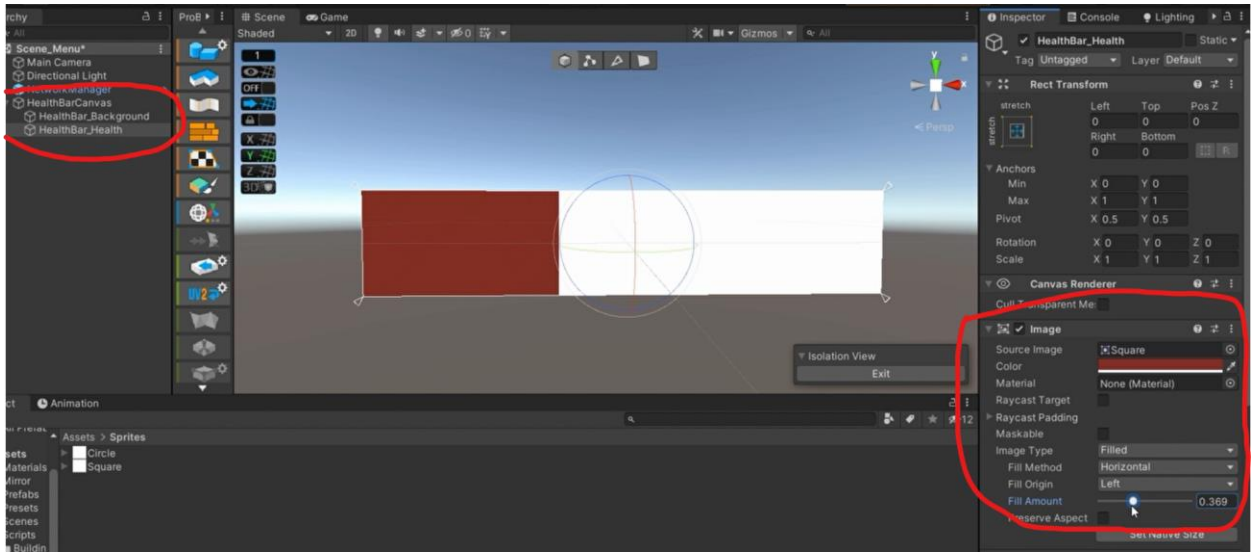
```
public class HealthDisplay : MonoBehaviour
{
  [SerializeField] private Health health = null;
  [SerializeField] private GameObject healthBarParent = null;
  [SerializeField] private Image healthBarImage = null;
  private void Awake()
  {
    health.ClientOnHealthUpdated += HandleHealthUpdated;
  }
  private void OnDestroy()
  {
    health.ClientOnHealthUpdated -= HandleHealthUpdated;
  }
  private void OnMouseEnter()
  {
    healthBarParent.SetActive(true);
  }
  private void OnMouseExit()
  {
    healthBarParent.SetActive(false);
  }
  private void HandleHealthUpdated(int currentHealth, int maxHealth)
  {
    healthBarImage.fillAmount = (float)currentHealth / maxHealth;
  }
}
```

حال که این مولفه را ساختیم باید مولفه سلامتی را تغییر داده تا زمان تغییر سلامتی HandleHealthUpdated را صدا بزند. برای این کار مقدار حال سلامتی را به متدی به نام HandleHealthUpdated در خود کد سلامتی hook می‌کنیم تا زمان تغییر صدا زده شود، سپس در متد

HandleHealthUpdated مقدار جدید را با مقدار قبلی جایگزین می‌کنیم تا صدا زده شده و کد HealthDisplay آن را نمایش دهد.

حال یک شی Canvas در یونیتی ساخته و در آن یک عکس عقب و یک عکس برای جلو که نشان دهنده سلامتی است می‌سازیم و Image Type را به حالت Filled در می‌آوریم تا با مقدار اعشاری داخل کد سازگاری داشته باشد. این شی را با مولفه‌های آن در زیر نشان داده‌ایم.

سپس این شی را داخل هر شی دارای سلامتی قرار می‌دهیم و در هر شی یک مولفه HealthDisplay قرار می‌دهیم و اجسام لازم را داخل آن قرار می‌دهیم.



شکل ۱۹) شی رابط کاربری میزان سلامتی

حال باید نابود شدن ساختمان‌ها و نیروها رو اضافه کنیم تا زمانی که سلامتی آنها به صفر می‌رسد نابود شده و از لیست شی‌های بازیکن پاک شود.

برای این کار ابتدا تغییراتی در UnitSpawner که ساختمان ماست انجام دهیم. سه متد باید اضافه شود. ۱ـ OnStartServer که متد ServerHandleDie را به عنوان اکشن به ServerOnDie مولفه سلامتی اضافه می‌کند تا زمانی که سلامتی به صفر رسید صدا زده شود. ۲- OnStopServer که متد ServerHandleDie را از ServerOnDie کم می‌کند. ۳- ServerHandleDie که شی ما که همان ساختمان باشد را از سرور و صفحه بازی پاک می‌کند.

تغییرات کد UnitSpawner را در زیر مشاهده می‌کنیم.

```
public class UnitSpawner : NetworkBehaviour, IPointerClickHandler
{
   [SerializeField] private Health health = null;
   [SerializeField] private GameObject unitPrefab = null;
   [SerializeField] private Transform unitSpawnPoint = null;
   #region Server
   public override void OnStartServer()
   {
      health.ServerOnDie += ServerHandleDie;
   }
   public override void OnStopServer()
   {
      health.ServerOnDie -= ServerHandleDie;
   }
   [Server]
   private void ServerHandleDie()
   {
      NetworkServer.Destroy(gameObject);
   }
   [Command]
   private void CmdSpawnUnit()
   {
      GameObject unitInstance = Instantiate(
         unitPrefab,
         unitSpawnPoint.position,
         unitSpawnPoint.rotation);
      NetworkServer.Spawn(unitInstance, connectionToClient);
   }
   #endregion
```

در مولفه Unit نیز که همان نیرو ماست نیز همین تغییرات بالا را انجام می‌دهیم که در زیر کد تغییر یافته آن را مشاهده می‌کنیم.

```
public class Unit : NetworkBehaviour
{
   [SerializeField] private Health health = null;
   [SerializeField] private UnitMovement unitMovement = null;
   [SerializeField] private Targeter targeter = null;
   [SerializeField] private UnityEvent onSelected = null;
   [SerializeField] private UnityEvent onDeselected = null;
   public static event Action<Unit> ServerOnUnitSpawned;
   public static event Action<Unit> ServerOnUnitDespawned;
   public static event Action<Unit> AuthorityOnUnitSpawned;
   public static event Action<Unit> AuthorityOnUnitDespawned;
   public UnitMovement GetUnitMovement()
   {
      return unitMovement;
```

```
    }
    public Targeter GetTargeter()
    {
        return targeter;
    }
    #region Server
    public override void OnStartServer()
    {
        ServerOnUnitSpawned?.Invoke(this);
        health.ServerOnDie += ServerHandleDie;
    }
    public override void OnStopServer()
    {
        health.ServerOnDie -= ServerHandleDie;
        ServerOnUnitDespawned?.Invoke(this);
    }
    [Server]
    private void ServerHandleDie()
    {
        NetworkServer.Destroy(gameObject);
    }
    #endregion
```

حال برای آنکه اگر نیرویی در انتخاب بود و نابود شد هنوز در حالت انتخاب شده نباشد زیرا لیست دارای یک متغییر null است باید آن نیرو را از لیست حذف کنیم تا اروری به وجود نیاید، که تغییرات زیر نیازمند است.

به Start باید متد AuthorityHandleDespawned را به اکشن AuthorityOnUnitDespawned اضافه کرد تا زمان نابود شدن نیرو AuthorityHandleDespawned صدا زده شود. باید متد OnDestroy را نیز اضافه کرد و متد AuthorityHandleDespawned را از اکشن AuthorityOnUnitDespawned کم کرد زیرا دیگر نیازی به آن نخواهد بود. در متد AuthorityHandleDespawned نیز نیرو مربوطه که نابود شده است را از لیست نیروهای انتخاب شده حذف می‌کنیم تا مشکلی که بالاتر گفتیم ایجاد نشود.

```
public class UnitSelectionHandler : MonoBehaviour
{
    [SerializeField] private RectTransform unitSelectionArea = null;
    [SerializeField] private LayerMask layerMask = new LayerMask();
    private Vector2 startPosition;
    private RTSPlayer player;
    private Camera mainCamera;
    public List<Unit> SelectedUnits { get; } = new List<Unit>();
    private void Start()
    {
```

```
        mainCamera = Camera.main;
        Unit.AuthorityOnUnitDespawned += AuthorityHandleUnitDespawned;
    }
    private void OnDestroy()
    {
        Unit.AuthorityOnUnitDespawned -= AuthorityHandleUnitDespawned;
    }
    private void AuthorityHandleUnitDespawned(Unit unit)
    {
        SelectedUnits.Remove(unit);
    }
```

برای برد و باخت در یک بازی استراتژی بازیکن باید قلعه اصلی یا مرکز فرماندهی رقیب خود را نابود کند پس ما نیاز به یک ساختمان که مرکز فرماندهی باشد داریم برای این کار یک مولفه مانند UnitSpawner باید بسازیم که نیرویی نمی‌سازد ولی نابودی آن موجب باخت بازیکن می‌شود.

ابتدا دو اکشن ServerOnBaseSpawned و ServerOnBaseDespawned را می‌سازیم تا در آینده به کمک آن سیستم برد و باخت را بسازیم. در متد OnStartServer متد ServerHandleDie را به متد ServerOnDie سلامتی اضافه می‌کنیم تا هنگام صفر شدن سلامتی این ساختمان متد ما صدا زده شود. در متد OnStopServer نیز متد ServerHandleDie را حذف می‌کنیم تا مشکلی در آینده پیش نیاید. در متد ServerHandleDie نیز این ساختمان را از سرور پاک می‌کنیم که زمانی صدا زده می‌شود که سلامتی ساختمان صفر شود. کد این مولفه در زیر آورده شده است.

```
public class UnitBase : NetworkBehaviour
{
    [SerializeField] private Health health = null;
    public static event Action<UnitBase> ServerOnBaseSpawned;
    public static event Action<UnitBase> ServerOnBaseDespawned;
    #region Server
    public override void OnStartServer()
    {
        health.ServerOnDie += ServerHandleDie;
        ServerOnBaseSpawned?.Invoke(this);
    }
    public override void OnStopServer()
    {
        ServerOnBaseDespawned?.Invoke(this);
        health.ServerOnDie -= ServerHandleDie;
    }
    [Server]
    private void ServerHandleDie()
    {
        NetworkServer.Destroy(gameObject);
    }
```

```
    #endregion
    #region Client
    #endregion
}
```

حال باید یک شی و یک مولفه برای سیستم برد و باخت درست کنیم در این مولفه که کد آن را در زیر مشاهده می‌کنیم ابتدا یک لیست از تمام مرکز فرماندهی‌ها که شی اصلی هر بازیکن است و با اضافه شدن بازیکن به مسابقه اضافه می‌شود قرار می‌دهیم.

متد ServerHandleBaseSpawned را می‌سازیم که هر بار که بازیکنی اضافه می‌شود یک مرکز فرماندهی به آن اضافه می‌شود. متد ServerHandleDespawned ار می‌سازیم که زمانی صدا زده می‌شود که یک مرکز فرماندهی نابود شده است، مرکز فرماندهی را از لیست حذف کرده و چک می‌کند تا اگر تعداد مرکز فرماندهی یک باشد پایان بازی را اعلام کند. در OnStartServer و OnStopServer دو متد بالا را به عنوان یک اکشن به ServerOnBaseSpawned و ServerOnBaseDespawned اضافه می‌کند. کد این مولفه را در زیر مشاهده می‌کنیم.

```
public class GameOverHandler : NetworkBehaviour
{
    private List<UnitBase> bases = new List<UnitBase>();
    #region Server
    public override void OnStartServer()
    {
        UnitBase.ServerOnBaseSpawned += ServerHandleBaseSpawned;
        UnitBase.ServerOnBaseDespawned += ServerHandleBaseDespawned;
    }
    public override void OnStopServer()
    {
        UnitBase.ServerOnBaseSpawned -= ServerHandleBaseSpawned;
        UnitBase.ServerOnBaseDespawned -= ServerHandleBaseDespawned;
    }
    [Server]
    private void ServerHandleBaseSpawned(UnitBase unitBase)
    {
        bases.Add(unitBase);
    }
    [Server]
    private void ServerHandleBaseDespawned(UnitBase unitBase)
    {
        bases.Remove(unitBase);
        if (bases.Count != 1) { return; }
        Debug.Log("Game Over");
    }
```

```
    #endregion
    #region Client
    #endregion
}
```

تا به این لحظه اتمام بازی به صورت یک log نمایش داده می‌شد حال می‌خواهیم آن را به صورت رابط کاربری نشان بدهیم.

برای این کار ابتدا در مولفه GameOverHandler یک اکشن نام ClientOnGameOver اضافه می‌کنیم سپس به جای Debug.Log در ServerHandleBaseDespawned شماره بازیکن باقی مانده در مسابقه را که همان برنده است سیو کرده و متد RpcGameOver را ساخته و اکشن ClientOnGameOver صدا می‌زنیم. تغییرات ذکر شده را در زیر مشاهده می‌کنیم.

```
public class GameOverHandler : NetworkBehaviour
{
    public static event Action<string> ClientOnGameOver;
    private List<UnitBase> bases = new List<UnitBase>();
    #region Server
    public override void OnStartServer()
    {
        UnitBase.ServerOnBaseSpawned += ServerHandleBaseSpawned;
        UnitBase.ServerOnBaseDespawned += ServerHandleBaseDespawned;
    }
    public override void OnStopServer()
    {
        UnitBase.ServerOnBaseSpawned -= ServerHandleBaseSpawned;
        UnitBase.ServerOnBaseDespawned -= ServerHandleBaseDespawned;
    }
    [Server]
    private void ServerHandleBaseSpawned(UnitBase unitBase)
    {
        bases.Add(unitBase);
    }
    [Server]
    private void ServerHandleBaseDespawned(UnitBase unitBase)
    {
        bases.Remove(unitBase);
        if (bases.Count != 1) { return; }
        int playerId = bases[0].connectionToClient.connectionId;
        RpcGameOver($"Player {playerId}");
    }
    #endregion
    #region Client
    [ClientRpc]
    private void RpcGameOver(string winner)
    {
```

```
    ClientOnGameOver?.Invoke(winner);
  }
  #endregion
}
```

سپس مولفه جدیدی به نام GameOverDisplay برای ساختن شی ای با این نام درست می‌کنیم که در این مولفه در یونیتی یک شی به نام gameOverDisplayParent که شی والد متن پیروزی است و شی متنی winnerNameText را قرار می‌دهیم. سپس به اکشن ClientOnGameOver که در بالاتر ذکر کردیم متد ClientHandleGameOver را اضافه کرده که در این متد متن شی winnerNameText را به اسم برنده تغییر می‌دهیم و شی والد را نمایش می‌دهیم. یک متد که برای دکمه‌ای که در زیر متن برنده مسابقه نمایش داده می‌شود نیز میسازیم با نام LeaveGame که با کلیک روی آن بازیکنان به صفحه اصلی برمی‌گردند. در این متد اگر بازیکن سرور بوده باشد با ترک آن سرور را می‌بندیم و تمام بازیکنان را به صفحه اصلی می‌فرستیم و اگر بازیکن یکی از متصلین به سرور باشد تنها او را به صفحه اصلی هدایت می‌کنیم. در زیر این مولفه را مشاهده می‌کنیم.

```
public class GameOverDisplay : MonoBehaviour
{
  [SerializeField] private GameObject gameOverDisplayParent = null;
  [SerializeField] private TMP_Text winnerNameText = null;
  private void Start()
  {
    GameOverHandler.ClientOnGameOver += ClientHandleGameOver;
  }
  private void OnDestroy()
  {
    GameOverHandler.ClientOnGameOver -= ClientHandleGameOver;
  }
  public void LeaveGame()
  {
    if (NetworkServer.active && NetworkClient.isConnected)
    {
      NetworkManager.singleton.StopHost();
    }
    else
    {
      NetworkManager.singleton.StopClient();
    }
  }
  private void ClientHandleGameOver(string winner)
  {
    winnerNameText.text = $"{winner} Has Won!";
    gameOverDisplayParent.SetActive(true);
  }
}
```

در زیر همچنین عکس از دکمه LeaveGame و متن پیروزی می‌بینیم.

شکل ۲۰) صفحه پایانی بازی و متن پیروزی بازیکن غالب

از آنجایی که شی HealthDisplay ما که سلامتی نیروها و ساختمان‌ها را نشان می‌دهد یک رابط کاربری است و در دو بعد است همیشه به سمت دوربین نیست زیرا با خود نیروها و ساختمان‌ها میچرخد پس برای اینکه همیشه به سمت دوربین باشد به مولفه زیر نیاز داریم و آن را در شی والد HealthDisplay قرار می‌دهیم.

کد آن را در زیر می‌بینیم که در آن با استفاده از transform.LookAt می‌توان شی را به سمت یک شی دیگر چرخاند که این کار را انجام داده‌ایم.

```
public class FaceCamera : MonoBehaviour
{
   private Transform mainCameraTransform;
   private void Start()
   {
      mainCameraTransform = Camera.main.transform;
   }
   private void LateUpdate()
   {
      transform.LookAt(
         transform.position + mainCameraTransform.rotation * Vector3.forward,
         mainCameraTransform.rotation * Vector3.up);
   }
}
```

از آنجایی که تمام نیروها از ساختمان‌های مختلف ساخته می‌شوند و خود ساختمان‌ها باید ابتدا ساخته شده و مانند مرکز فرماندهی از ابتدا وجود ندارند مولفه‌ای به نام ساختمان ساخته و در آن یک آیکون برای قرار دادن در رابط کاربری برای انتخا و ساخت، یک آیدی برای شناسایی و خاص کردن هر ساختمان، یک قیمت برای هر ساختمان تا بازیکن با پرداخت آن بتواند ساختمان را بسازد.

سپس برای گرفتن هر یک از این سه متغییر در یک مولفه دیگر یک متد می‌سازیم. سپس چار اکشن ساخت و نابودی ساختمان در سرور و ساخت و نابودی ساختمان برای بازیکن می‌سازیم و چهار متد برای صدا زدن این اکشن‌ها از یک مولفه دیگر می‌سازیم. کد این مولفه را در زیر می‌بینیم.

```
public class Building : NetworkBehaviour
{
   [SerializeField] private Sprite icon = null;
   [SerializeField] private int id = -1;
   [SerializeField] private int price = 100;
   public static event Action<Building> ServerOnBuildingSpawned;
   public static event Action<Building> ServerOnBuildingDespawned;
   public static event Action<Building> AuthorityOnBuildingSpawned;
   public static event Action<Building> AuthorityOnBuildingDespawned;
   public Sprite GetIcon()
   {
      return icon;
   }
   public int GetId()
   {
      return id;
   }
   public int GetPrice()
   {
      return price;
   }
   #region Server
   public override void OnStartServer()
   {
      ServerOnBuildingSpawned?.Invoke(this);
   }
   public override void OnStopServer()
   {
      ServerOnBuildingDespawned?.Invoke(this);
   }
   #endregion
   #region Client
   public override void OnStartAuthority()
   {
      AuthorityOnBuildingSpawned?.Invoke(this);
   }
   public override void OnStopClient()
   {
      if (!hasAuthority) { return; }
      AuthorityOnBuildingDespawned?.Invoke(this);
   }
   #endregion
}
```

سپس در مولفه RTSPlayer همانطور که برای نیروها یک لیست و متدها و اکشن‌های مورد نیاز آن برای اضافه و کم کردن نیروها از لیست نوشتیم برای ساختمان نیز این لیست، متد و اکشن‌ها را اضافه می‌کنیم. در زیر کدهای اضافه شده به کد RTSPlayer را آورده‌ایم.

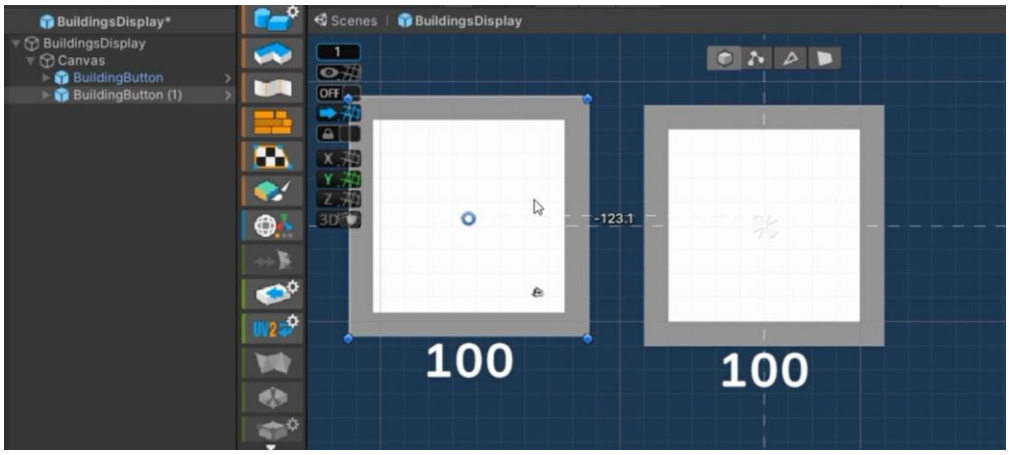
```
public class RTSPlayer : NetworkBehaviour
{
    private List<Unit> myUnits = new List<Unit>();
    private List<Building> myBuildings = new List<Building>();
    public List<Building> GetMyBuildings()
    {
        return myBuildings;
    }
    #region Server
    public override void OnStartServer()
    {
        Building.ServerOnBuildingSpawned += ServerHandleBuildingSpawned;
        Building.ServerOnBuildingDespawned += ServerHandleBuildingDespawned;
    }
    public override void OnStopServer()
    {
        Building.ServerOnBuildingSpawned -= ServerHandleBuildingSpawned;
        Building.ServerOnBuildingDespawned -= ServerHandleBuildingDespawned;
    }
    private void ServerHandleBuildingSpawned(Building building)
    {
        if (building.connectionToClient.connectionId != connectionToClient.connectionId) { return; }
        myBuildings.Add(building);
    }
    private void ServerHandleBuildingDespawned(Building building)
    {
        if (building.connectionToClient.connectionId != connectionToClient.connectionId) { return; }
        myBuildings.Remove(building);
    }
    #endregion
    #region Client
    public override void OnStartAuthority()
    {
        Building.AuthorityOnBuildingSpawned += AuthorityHandleBuildingSpawned;
        Building.AuthorityOnBuildingDespawned += AuthorityHandleBuildingDespawned;
    }
    public override void OnStopClient()
    {
        Building.AuthorityOnBuildingSpawned -= AuthorityHandleBuildingSpawned;
        Building.AuthorityOnBuildingDespawned -= AuthorityHandleBuildingDespawned;
    }
    private void AuthorityHandleBuildingSpawned(Building building)
    {
        myBuildings.Add(building);
    }
```

```
    private void AuthorityHandleBuildingDespawned(Building building)
    {
        myBuildings.Remove(building);
    }
    #endregion
}
```

حال برای ساخت ساختمان‌ها نیاز به رابط کاربری و دکمه‌هایی روی صفحه داریم تا آن‌ها را انتخاب کرده و روی صفحه کشیده و جایگزین کنیم. ابتدا یک دکمه که شامل عکس پس ضمینه دکمه است و در داخل آن یک عکس برای مشخص کردن اینکه کدام یک از ساختمان‌ها درحال ساخت است و در زیر آن یک متن برای قیمت می‌سازیم که به صورت زیر می‌شود.

شکل ۲۱) دکمه ساخت ساختمان همراه متن قیمت ساختمان

سپس یک شی در صفحه ساخته و داخل آن یک Canvas اضافه کرده و در داخل آن می‌توانیم دکمه‌هایی که بالاتر ساختیم را قرار دهیم. تصویری از آن در زیر نمایش داده شده است.



شکل ۲۲) رابط کاربری ساخت ساختمان‌ها

وقتی در حال انتخاب و کشیدن ساختمان برای پیدا کردن مکان مناسب برای آن هستیم نیاز به نمایش یک پیش نمایش از ساختمان داریم که باید آن را به کد ساختمان اضافه کنیم. در زیر کد اضافه شده نشان داده می‌شود.

```
public class Building : NetworkBehaviour
{
  [SerializeField] private GameObject buildingPreview = null;
  [SerializeField] private Sprite icon = null;
  [SerializeField] private int id = -1;
  [SerializeField] private int price = 100;
  public static event Action<Building> AuthorityOnBuildingSpawned;
  public static event Action<Building> AuthorityOnBuildingDespawned;
  public GameObject GetBuildingPreview()
  {
    return buildingPreview;
  }
```

حال به ساختن مولفه‌ای برای دکمه‌هایی که بالاتر ساختیم می‌پردازیم شیوه کار دکمه‌ها به این صورت است که بر روی آنها کلیک شده و سپس موس به داخل صفحه حرکت داده شده که در این مرحله پیش نمایش ساختمان نمایش داده می‌شود و پس از آنکه بازیکن محل مناسب را برای ساخت آن پیدا کرد موس را رها کرده و ساختمان ساخته می‌شود. برای این‌کار دکمه باید شامل ساختمان، یک عکس به عنوان آیکون ساختمان، یک متن به عنوان برچسب قیمت و یک LayerMask که نشان می‌دهد پیش نمایش و خود ساختمان روی چه سطحی قابل ساخت و نمایش است. در ابتدا دوربین را پیدا کرده و پس از آن آیکون و قیمت ساختمان را از مولفه ساختمان گرفته و نمایش می‌دهیم. در متدی که هر فریم صدا زده می‌شود اگر بازکین را نمی‌شناسیم آن را پیدا کرده و سپس اگر پیش نمایش ساختمان در حال نمایش است آن را آپدیت می‌کنیم. در متد OnPointerDown اگر کلیک مورد نظر کلیک چپ بود پیش نمایش ساختمان را در صفحه تولید کرده ولی آن را چون نمی‌خواهیم هر جای صفحه نمایش دهیم خاموش می‌کنیم. در متد UpdateBuildingPreview مکان موس را خوانده و اگر مکان آن روی سطح قابل ساخت بود پیش نمایش ساختمان را نمایش می‌دهیم. در متد OnPointerUp اگر پیش نمایشی از ساختمان وجود داشت مکان موس را خوانده و به کمک یک دستور که در آینده در کد RTSPlayer اضافه می‌کنیم سعی به ساخت ساختمان در مکان مشخص شده می‌کنیم و پیش نمایش آن را نابود می‌کنیم.

در زیر کد این مولفه را مشاهده می‌کنیم.

```
public class BuildingButton : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
```

```
{
    [SerializeField] private Building building = null;
    [SerializeField] private Image iconImage = null;
    [SerializeField] private TMP_Text priceText = null;
    [SerializeField] private LayerMask floorMask = new LayerMask();
    private Camera mainCamera;
    private RTSPlayer player;
    private GameObject buildingPreviewInstance;
    private Renderer buildingRendererInstance;
    private void Start()
    {
        mainCamera = Camera.main;
        iconImage.sprite = building.GetIcon();
        priceText.text = building.GetPrice().ToString();
    }
    private void Update()
    {
        if (player == null)
        {
            player = NetworkClient.connection.identity.GetComponent<RTSPlayer>();
        }
        if (buildingPreviewInstance == null) { return; }
        UpdateBuildingPreview();
    }
    public void OnPointerDown(PointerEventData eventData)
    {
        if (eventData.button != PointerEventData.InputButton.Left) { return; }
        buildingPreviewInstance = Instantiate(building.GetBuildingPreview());
        buildingRendererInstance = buildingPreviewInstance.GetComponentInChildren<Renderer>();
        buildingPreviewInstance.SetActive(false);
    }
    public void OnPointerUp(PointerEventData eventData)
    {
        if (buildingPreviewInstance == null) { return; }
        Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());
        if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, floorMask))
        {
        player.CmdTryPlaceBuilding(building.GetId(), hit.point);
        }
        Destroy(buildingPreviewInstance);
    }
    private void UpdateBuildingPreview()
    {
        Ray ray = mainCamera.ScreenPointToRay(Mouse.current.position.ReadValue());
        if (!Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, floorMask)) { return; }
        buildingPreviewInstance.transform.position = hit.point;
        if(!buildingPreviewInstance.activeSelf)
        {
            buildingPreviewInstance.SetActive(true);
        }
    }
```

}

حال به توضیح دستور CmdTryPlaceBuilding که در بالاتر گفتیم در مولفه RTSPlayer می‌پردازیم. در این دستور ساختمانی که قسط ساخت آن داریم را با استفاده از آیدی داده شده پیدا کرده و آن را در مکان داده شده ساخته و سپس در سرور آن را می‌سازیم و تعلق آن را به بازیکن نسبت می‌دهیم. این دستور اضافه شده را در زیر مشاهده می‌کنید.

```
[SerializeField] private Building[] buildings = new Building[0];
[Command]
public void CmdTryPlaceBuilding(int buildingId, Vector3 point)
{
    Building buildingToPlace = null;
    foreach (Building building in buildings)
    {
        if (building.GetId() == buildingId)
        {
            buildingToPlace = building;
            break;
        }
    }
    if (buildingToPlace == null) { return; }
    GameObject buildingInstance =
        Instantiate(buildingToPlace.gameObject, point, buildingToPlace.transform.rotation);
    NetworkServer.Spawn(buildingInstance, connectionToClient);
}
```

حال برای آنکه منابع هر بازیکن را ثبت کنیم داخل مولفه RTSPlayer یک عدد int به عنوان resources تعریف می‌کنیم و دو متد Get و Set برای آن اضافه می‌کنیم. پس از آن یک اکشن و یک متد از طرف موکل برای تغییر و عدد resources اضافه می‌کنیم. کدهای اضافه شده را در زیر مشاهده می‌کنیم.

```
public class RTSPlayer : NetworkBehaviour
{
    [SyncVar(hook = nameof(ClientHandleResourcesUpdated))]
    private int resources = 500;
    public event Action<int> ClientOnResourcesUpdated;
    public int GetResources()
    {
        return resources;
    }
    [Server]
    public void SetResources(int newResources)
    {
        resources = newResources;
    }
    private void ClientHandleResourcesUpdated(int oldResources, int newResources)
    {
```

```
        ClientOnResourcesUpdated?.Invoke(newResources);
    }
```

حال یک مولفه ResourcesDisplay برای نمایش منابع درست می‌کنیم. در این کد resourcesText یک متن برای نمایش منابع player یک مولفه بازیکن برای گرفتن مقدار منابع از آن است. در متدی که هرفریم اجرا می‌شود ابتدا بازیکن را یافته و متدی به نام ClientHandleResourcesUpdated که پایین‌تر توضیح می‌دهیم را به عنوان یک اکشن به متد ClientOnResourcesUpdated اضافه می‌کنیم. در OnDestroy متد بالا که به عنوان اکشن اضافه کرده بودیم را کم می‌کنیم. در متد ClientHandleResourcesUpdated متن resourcesText را با مقدار صحیح آپدیت می‌کنیم. کد این مولفه را در زیر مشاهده می‌کنیم.

```
public class ResourcesDisplay : MonoBehaviour
{
    [SerializeField] private TMP_Text resourcesText = null;
    private RTSPlayer player;
    private void Update()
    {
        if (player == null)
        {
            player = NetworkClient.connection.identity.GetComponent<RTSPlayer>();

            if(player != null)
            {
                ClientHandleResourcesUpdated(player.GetResources());
                player.ClientOnResourcesUpdated += ClientHandleResourcesUpdated;
            }
        }
    }
    private void OnDestroy()
    {
        player.ClientOnResourcesUpdated -= ClientHandleResourcesUpdated;
    }
    private void ClientHandleResourcesUpdated(int resources)
    {
        resourcesText.text = $"Resources: {resources}";
    }
}
```

حال باید یک مولفه ResourceGeneraor برای ساختمان‌های تولید منابع بسازیم که در پیش ساخته آن‌ها قرار داده تا بتوانیم آن‌ها را در زمان بازی ساخته و منابع خود را افزایش دهیم. در این مولفه یک متغییر مولفه سلامتی، یک عدد resourcesPerInterval برای که مشخص کننده مقدار اضافه شدن منابع به ازای هر دوره است، یک عدد interval برای مشخص کردن زمان هر دوره است.

در متد onStartServer تایمر را برابر دوره قرار می‌دهیم، سپس بازیکن را پیدا می‌کنیم، پس از آن به مولفه سلامتی و GameOverHandler دو متد ServerHandleDie که برای رسیدگی به عمل نابودی ساختمان و

ServerHandleGameOver برای خاموش کردن این مولفه در زمان باخت، را اضافه می‌کنیم. در متد OnStopServer همین دو متد بالا را کم می‌کنیم. در متدی که در هر فریم صدا زده می‌شود یک تایمر معکوس ساخته و هر زمان که تایمر به صفر رسید منابع بازیکن را به تعداد resourcesPerInterval اضافه می‌کنیم. کد این مولفه را در زیر می‌بینیم.

```
public class ResourceGenerator : NetworkBehaviour
{
   [SerializeField] private Health health = null;
   [SerializeField] private int resourcesPerInterval = 10;
   [SerializeField] private float interval = 2f;
   private float timer;
   private RTSPlayer player;
   public override void OnStartServer()
   {
      timer = interval;
      player = connectionToClient.identity.GetComponent<RTSPlayer>();
      health.ServerOnDie += ServerHandleDie;
      GameOverHandler.ServerOnGameOver += ServerHandleGameOver;
   }
   public override void OnStopServer()
   {
      health.ServerOnDie -= ServerHandleDie;
      GameOverHandler.ServerOnGameOver -= ServerHandleGameOver;
   }
   [ServerCallback]
   private void Update()
   {
      timer -= Time.deltaTime;
      if(timer <= 0)
      {
         player.SetResources(player.GetResources() + resourcesPerInterval);

         timer += interval;
      }
   }
   private void ServerHandleDie()
   {
      NetworkServer.Destroy(gameObject);
   }
   private void ServerHandleGameOver()
   {
      enabled = false;
   }
}
```

حال برای آنکه بازیکن نتواند تنها با یک کلیک به هر نیرویی رسیده و مجبور شود مدتی صبر کند تا آن نیرو تعلیم داده شود ولی همچنان مجبور نباشد تا برای سفارش هر نیرو به اندازه زمان ساخت آن نیرو صبر کند نیاز به مکانیزم صف ساخت نیرو داریم برای اینکار ابتدا تغییرات زیر را در کد مولفه UnitSpawner انجام

می‌دهیم. همچنین کم کردن منابع به ازای ساخت نیرو را هم اضافه می‌کنیم. ابتدا متغیر unitPrefab را به نوع Unit تغییر می‌دهیم، سپس متغییرهای زیر را اضافه می‌کنیم. یک متن برای تعداد باقی مانده نیروهای سفارش داده شده در صف remainingUnitsText اضافه می‌کنیم، یک عکس که با زمان باقی مانده ساخت را به شکل یک ساعت نشان دهد unitProgressImage، بیشترین تعداد نیرو در حال ساخت در صف maxUnitQueue، مسافتی که نیروهای ساخته شده در آن پخش می‌شود spawnMoveRange، زمان ساخت هر نیرو unitSpawnDuration

```
[SerializeField] private Unit unitPrefab = null;
[SerializeField] private TMP_Text remainingUnitsText = null;
[SerializeField] private Image unitProgressImage = null;
[SerializeField] private int maxUnitQueue = 5;
[SerializeField] private float spawnMoveRange = 7f;
[SerializeField] private float unitSpawnDuration = 5f;
```

در متدی که هر فریم صدا زده می‌شود از سمت سرور فرایند ساخت نیروها و از سمت بازیکن رابط کاربری تایمر صف ساخت را آپدیت می‌کنیم.

```
private void Update()
{
    if (isServer)
    {
        ProduceUnits();
    }

    if (isClient)
    {
        UpdateTimerDisplay();
    }
}
```

حال متد ProduceUnits که در بالا می‌بینیم را می‌سازیم. در این متد اگر تعداد نیروهای در صف بیشتر از صفر بود تایمر را راه انداخته و زمانی که تایمر به زمان مشخص رسید نیرو را تولید کرده و در سرور اضافه می‌کنیم سپس به نیرو دستور رفتن به محلی رندوم نزدیک ساختمان سازنده فرستاده و یک عدد از نیروهای در صف کم کرده و تایمر را صفر میکنیم.

این متد را در کد زیر می‌بینیم.

```
[Server]
private void ProduceUnits()
{
```

```
    if (queuedUnits == 0) { return; }
    unitTimer += Time.deltaTime;
    if (unitTimer < unitSpawnDuration) { return; }
    GameObject unitInstance = Instantiate(
        unitPrefab.gameObject,
        unitSpawnPoint.position,
        unitSpawnPoint.rotation);
    NetworkServer.Spawn(unitInstance, connectionToClient);
    Vector3 spawnOffset = Random.insideUnitSphere * spawnMoveRange;
    spawnOffset.y = unitSpawnPoint.position.y;
    UnitMovement unitMovement = unitInstance.GetComponent<UnitMovement>();
    unitMovement.ServerMove(unitSpawnPoint.position + spawnOffset);
    queuedUnits--;
    unitTimer = 0f;
}
```

ساخت نیرو قبل از این در متد CmdSpawnUnit بود که آن را به کد زیر تغییر داده و ساخت نیرو را به متد بالا سپرده‌ایم. در این متد اگر تعداد نیروهای داخل صف بیش از مقدار تعیین شده نبود، بازیکن را یافته و منابع آن را بررسی کرده و مقدار منابع لازم برای ساخت نیرو را کم کرده و یک نیرو به صف اضافه می‌کنیم. کد این متد را در زیر مشاهده می‌کنیم.

```
private void CmdSpawnUnit()
{
    GameObject unitInstance = Instantiate(
        unitPrefab,
        unitSpawnPoint.position,
        unitSpawnPoint.rotation);
    if (queuedUnits == maxUnitQueue) { return; }
    NetworkServer.Spawn(unitInstance, connectionToClient);
    RTSPlayer player = connectionToClient.identity.GetComponent<RTSPlayer>();
    if (player.GetResources() < unitPrefab.GetResourceCost()) { return; }

    queuedUnits++;
    player.SetResources(player.GetResources() - unitPrefab.GetResourceCost());
}
```

در متد UpdateTimerDisplay که در متد Update دیدیم قصد به روز رسانی فریم به فریم تایمر را داریم که در بالای هر ساختمان سازنده نیرو قرار دارد. این متد را در زیر مشاهده می‌کنیم.

```
private void UpdateTimerDisplay()
{
    float newProgress = unitTimer / unitSpawnDuration;

    if (newProgress < unitProgressImage.fillAmount)
    {
        unitProgressImage.fillAmount = newProgress;
    }
```

```
    else
    {
        unitProgressImage.fillAmount = Mathf.SmoothDamp(
            unitProgressImage.fillAmount,
            newProgress,
            ref progressImageVelocity,
            0.1f
        );
    }
}
```

در ساخت ساختمان در حال حاضر می‌توانیم آن را در هر مکانی که روی نقشه باشد بسازیم، ولی بهتر است تنها در مکان‌هایی قابل ساخت باشد که نزدیک ساختمان‌های دیگر باشد. همچنین ساختمان‌ها در حال حاضر قابل ساخت بر روی یک دیگر نیز هستند که اشتباه است. برای درست کردن آن به تغییرات زیر در کد مولفه RTSPlayer نیاز است. ابتدا در کد RTSPlayer دو متغییر buildingBlockLayer که شامل لایه‌هایی از صفحه شده که نمیتوان روی آن‌ها ساختمان را ساخت که شامل ساختمان‌های دیگر و نیروها است، buldingRangeLimit که شعاع دورترین فاصله ساخت یک ساختمان از ساختمان دیگر است.

```
[SerializeField] private LayerMask buildingBlockLayer = new LayerMask();
[SerializeField] private Building[] buildings = new Building[0];
[SerializeField] private float buildingRangeLimit = 5f;
```

در متد CanPlaceBuilding که می‌خواهیم تست کنیم آیا محل قابل ساخت است که ابتدا به تست Physics.CheckBox تست می‌کنیم آیا ساختمان با شی‌ای از لایه buildingBlockLayer برخورد می‌کند یا نه سپس در foreach تست می‌کنیم آیا محل ساخت در شعاع ساخت است و پس از آن اگر در شعاع بود و با شی دیگری برخورد نکرد متد true برمیگرداند.

```
public bool CanPlaceBuilding(BoxCollider buildingCollider, Vector3 point)
{
    if (Physics.CheckBox(
            point + buildingCollider.center,
            buildingCollider.size / 2,
            Quaternion.identity,
            buildingBlockLayer))
    {
        return false;
    }
    foreach (Building building in myBuildings)
    {
        if ((point - building.transform.position).sqrMagnitude
            <= buildingRangeLimit * buildingRangeLimit)
        {
            return true;
```

```
        }
    }
    return false;
}
```

سپس با تغییراتی در متد دستور CmdTryPlaceBuilding تست می‌کنیم آیا منابع کافی برای ساخت وجود دارد و آیا متد بالا true برمیگرداند سپس منابع لازم را کم کرده و ساختمان را می‌سازیم.

```
[Command]
public void CmdTryPlaceBuilding(int buildingId, Vector3 point)
{
    Building buildingToPlace = null;
    foreach (Building building in buildings)
    {
        if (building.GetId() == buildingId)
        {
            buildingToPlace = building;
            break;
        }
    }
    if (buildingToPlace == null) { return; }
    if (resources < buildingToPlace.GetPrice()) { return; }
    BoxCollider buildingCollider = buildingToPlace.GetComponent<BoxCollider>();
    if (!CanPlaceBuilding(buildingCollider, point)) { return; }
    GameObject buildingInstance =
        Instantiate(buildingToPlace.gameObject, point, buildingToPlace.transform.rotation);
    NetworkServer.Spawn(buildingInstance, connectionToClient);
    SetResources(resources - buildingToPlace.GetPrice());
}
```

در بازی‌های استراتژی به دلیل قاطی نشدن نیروها و ساختمان‌ها به هر بازیکن و متعلقات آن یک رنگ اختصاص داده می‌شود، برای این کار ابتدا تغییراتی در دو کد RTSPlayer و RTSNetworkManager می‌دهیم. ابتدا در RTSPlayer یک متغییر رنگ تعریف کرده و برای آن Getter و Setter تعریف می‌کنیم.

```
private Color teamColor = new Color();
public Color GetTeamColor()
{
    return teamColor;
}
[Server]
public void SetTeamColor(Color newTeamColor)
{
    teamColor = newTeamColor;
}
```

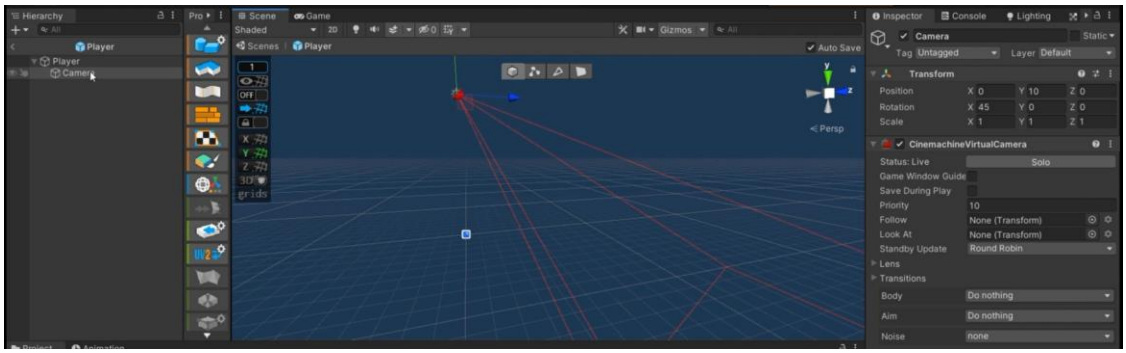
سپس در RTSNetworkManager به هر بازیکن پس از اضافه شدن آن به سرور یک رنگ جدید اختصاص می‌دهیم. کد کامل OnServerAddPlayer را پس از تغییر مشاهده می‌کنیم.

```
public override void OnServerAddPlayer(NetworkConnection conn)
{
    base.OnServerAddPlayer(conn);
    RTSPlayer player = conn.identity.GetComponent<RTSPlayer>();
    player.SetTeamColor(new Color(
        UnityEngine.Random.Range(0f, 1f),
        UnityEngine.Random.Range(0f, 1f),
        UnityEngine.Random.Range(0f, 1f)
    ));
    GameObject unitSpawnerInstance = Instantiate(
        unitSpawnerPrefab,
        conn.identity.transform.position,
        conn.identity.transform.rotation);
    NetworkServer.Spawn(unitSpawnerInstance, conn);
}
```

حال یک مولفه برای برقرار کردن رنگ روی تمام متعلقات بازیکن نیاز داریم که در هر شی که نیاز به رنگ دارد قرار گرفته و زیر شی‌هایی که داری material است را رنگ می‌کند. کد این مولفه را در زیر مشاهده می‌کنیم.

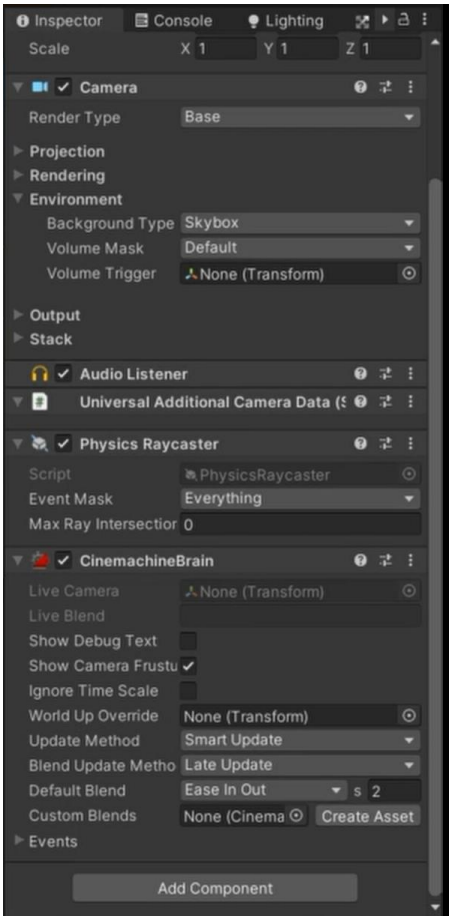
```
public class TeamColorSetter : NetworkBehaviour
{
    [SerializeField] private Renderer[] colorRenderers = new Renderer[0];

    [SyncVar(hook = nameof(HandleTeamColorUpdated))]
    private Color teamColor = new Color();
    #region Server
    public override void OnStartServer()
    {
        RTSPlayer player = connectionToClient.identity.GetComponent<RTSPlayer>();
        teamColor = player.GetTeamColor();
    }
    #endregion
    #region Client
    private void HandleTeamColorUpdated(Color oldColor, Color newColor)
    {
        foreach(Renderer renderer in colorRenderers)
        {
            renderer.material.SetColor("_BaseColor", newColor);
        }
    }
    #endregion
}
```

از آنجایی که معمولا نقشه‌های یک بازی استراتژی پهناور است نیاز به حرکت دوربین داریم برای این‌کار ابتدا طبق شکل زیر یک شی CinemachineVirtualCamera به پیش ساخته بازیکن اضافه کرده و مکان کلی آن را تنظیم می‌کنیم.



شکل ۲۳) شی دوربین در پیش ساخته بازیکن

پس از آن به دوربین اصلی داخل صفحه یک CinemachineBrain همچون شکل زیر اضافه می‌کنیم.



شکل ۲٤) مولفه CinemachineBrain در یونیتی

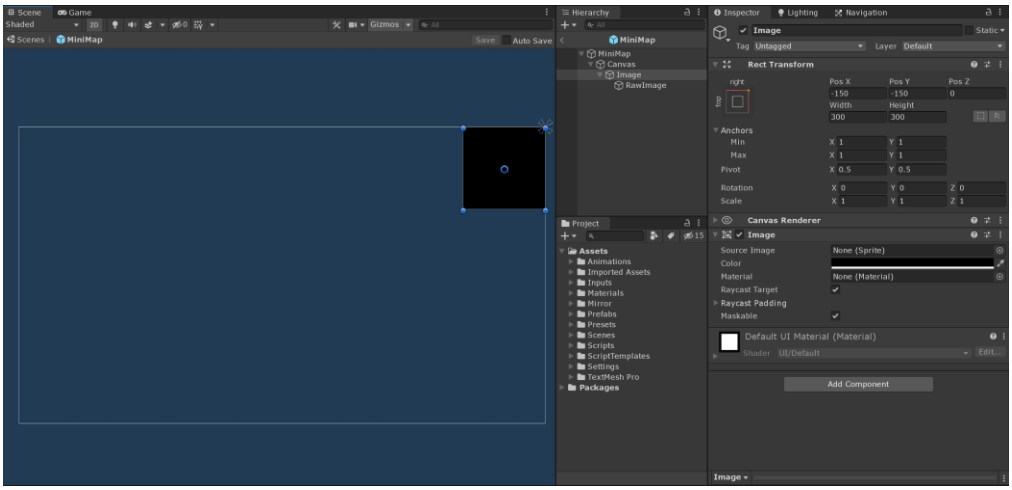
حال یک مولفه برای کنترل دوربین ساخته و در داخل پیش ساخته بازیکن قرار می‌دهیم. کد آن را در زیر توضیح می‌دهیم. در متد OnStartAuthority که بازیکن شروع به بازی می‌کند دوربین را روشن کرده و سپس اکشن SetPreviousInput را به متدهای حرکت دوربین اضافه می‌کنیم. در متد Update اگر دوربین متعلق به بازیکن بود و بازیکن در بازی بوده و روی برنامه دیگری توجه نداشت مکان دوربین را آپدیت می‌کنیم. در متد UpdateCameraPosition مکان حال دوربین را ذخیره کرده سپس اگر مکان موس از راست ، چپ ، بالا و پایین گوشه صفحه بود دوربین را به سمت آن حرکت می‌دهیم. همچنین استفاده از دکمه‌های فلش کیبورد و یا WASD نیر همین کارایی را دارند.

کد این مولفه را در زیر می‌بینیم.

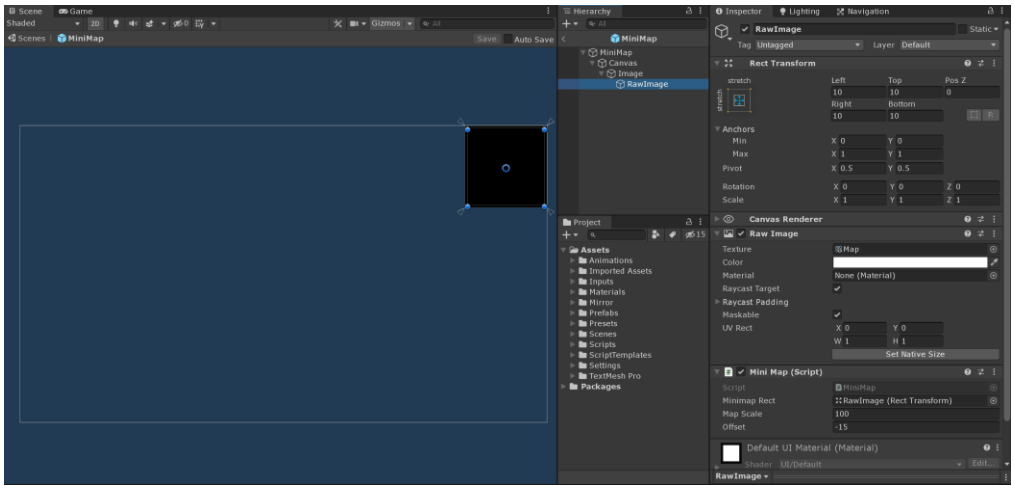
```
public class CameraController : NetworkBehaviour
{
   [SerializeField] private Transform playerCameraTransform = null;
   [SerializeField] private float speed = 20f;
   [SerializeField] private float screenBorderThickness = 10f;
   [SerializeField] private Vector2 screenXLimits = Vector2.zero;
   [SerializeField] private Vector2 screenZLimits = Vector2.zero;
   private Vector2 previousInput;
   private Controls controls;
   public override void OnStartAuthority()
   {
      playerCameraTransform.gameObject.SetActive(true);
      controls = new Controls();
      controls.Player.MoveCamera.performed += SetPreviousInput;
      controls.Player.MoveCamera.canceled += SetPreviousInput;
      controls.Enable();
   }
   [ClientCallback]
   private void Update()
   {
      if (!hasAuthority || !Application.isFocused) { return; }
      UpdateCameraPosition();
   }
   private void UpdateCameraPosition()
   {
      Vector3 pos = playerCameraTransform.position;
      if (previousInput == Vector2.zero)
      {
         Vector3 cursorMovement = Vector3.zero;
         Vector2 cursorPosition = Mouse.current.position.ReadValue();
         if (cursorPosition.y >= Screen.height - screenBorderThickness)
         {
            cursorMovement.z += 1;
         }
         else if (cursorPosition.y <= screenBorderThickness)
         {
```

```
            cursorMovement.z -= 1;
        }
        if (cursorPosition.x >= Screen.width - screenBorderThickness)
        {
            cursorMovement.x += 1;
        }
        else if (cursorPosition.x <= screenBorderThickness)
        {
            cursorMovement.x -= 1;
        }
        pos += cursorMovement.normalized * speed * Time.deltaTime;
    }
    else
    {
        pos += new Vector3(previousInput.x, 0f, previousInput.y) * speed * Time.deltaTime;
    }
    pos.x = Mathf.Clamp(pos.x, screenXLimits.x, screenXLimits.y);
    pos.z = Mathf.Clamp(pos.z, screenZLimits.x, screenZLimits.y);
    playerCameraTransform.position = pos;
  }
  private void SetPreviousInput(InputAction.CallbackContext ctx)
  {
    previousInput = ctx.ReadValue<Vector2>();
  }
}
```

در بازی‌های استراتژی معمولا نقشه‌های کوچکی کنار صفحه موجود است که در آن نیروها و ساختمان‌های بازیکنان به نمایش گذاشته می‌شود، برای ساخت این ویژگی ابتدا یک شی که داخل آن یک Canvas قرار می‌دهیم، سپس یک عکس برای بک‌گراند نقشه و سپس یک RawImage در داخل آن قرار می‌دهیم که همان نقشه است در زیر این شی را می‌بینیم.

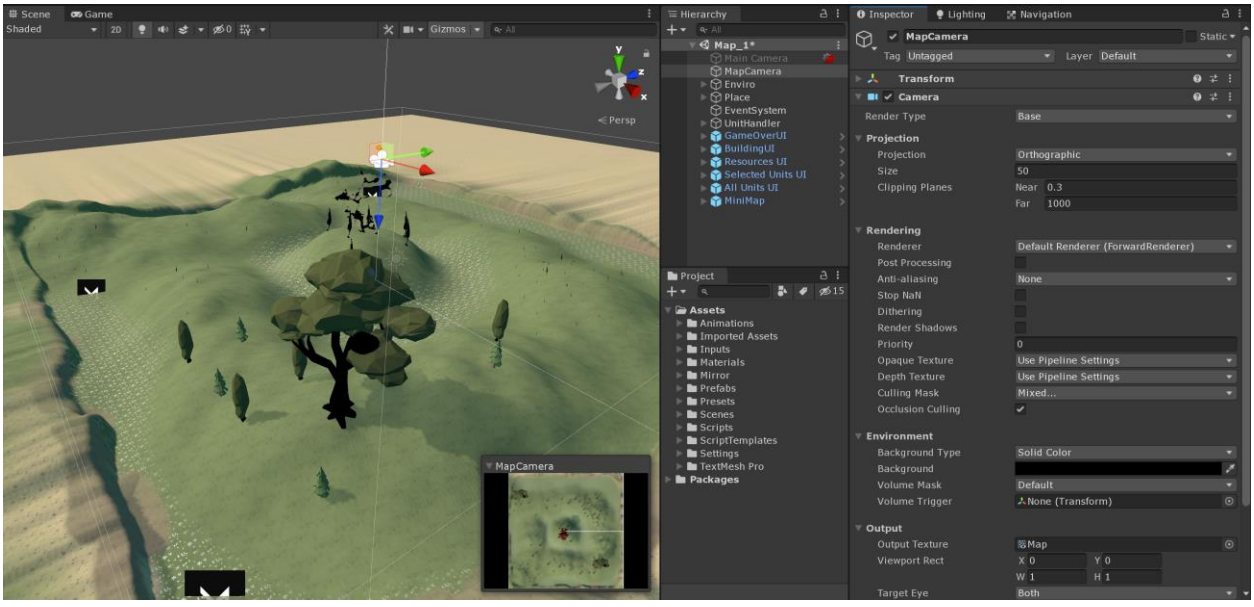


شکل ۲۵) شی نقشه کوچک و مولفه‌های آن



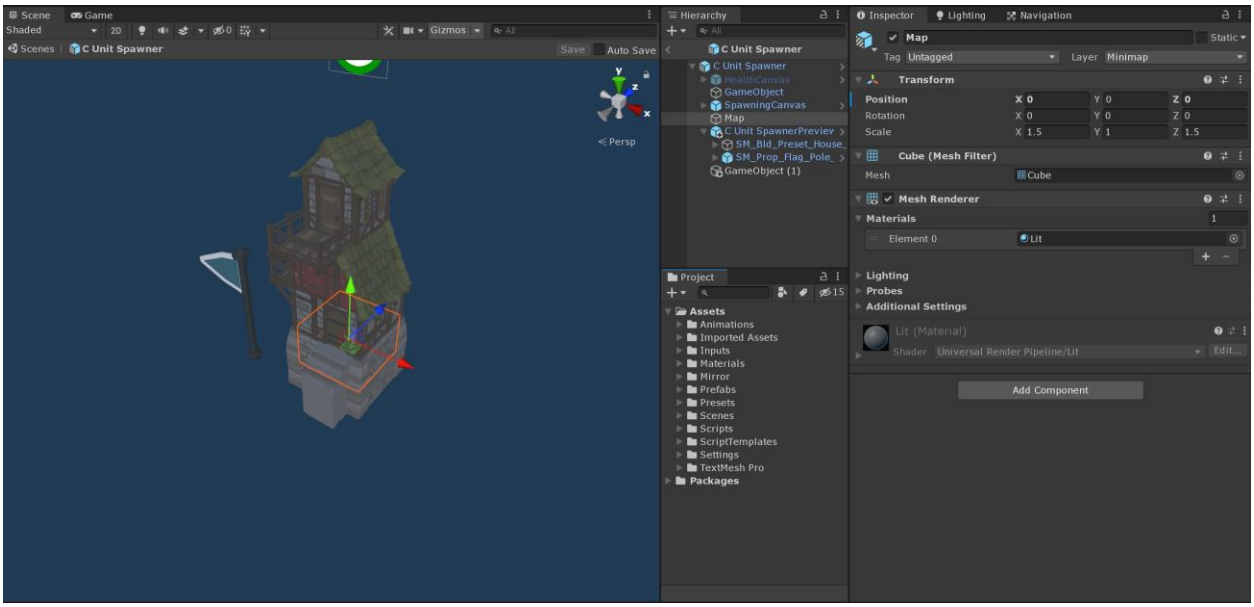
شکل ۲۶) شی نقشه کوچک و اضافه شدن مولفه Mini Map

پس از آن یک دوربین جدید در صفحه بازی قرار می‌دهیم و آن را بالا به سمت کل نقشه قرار می‌دهیم، سپس این دوربین را روی حالت orthographic قرار داده و به اندازه طول زمین به آن Size می‌دهیم. این دوربین را در زیر می‌بینیم.



شکل ۲۷) نقشه کلی بازی و محل قرارگیری دوربین نقشه کوچک

حال اجسامی که می‌خواهیم در نقشه دیده شوند را یک زیر شی با شکل دلخواه اضافه کرده و آن را در لایه MiniMap قرار می‌دهیم و به دوربین قابلیت دیدن این لایه را می‌دهیم. این زیر شی را در شکل زیر می‌بینیم.



شکل ۲۸) شی ساختمان و اضافه کردن شکلی دلخواه برای نمایش در نقشه کوچک

شکل نهایی نقشه را به همراه متعلقات بازیکن نمایش می‌دهد که در زیر آورده شده است.

شکل ۲۹) شکل نهایی نقشه کوچک و نمایش دادن متعلقات بازیکن

حال که طرح کلی نقشه کوچک را درست کرده‌ایم مولفه‌ای برای پیمودن نقشه به کلیک روی محل دلخواه در نقشه و بردن دوربین به آن درست می‌کنیم. کد این مولفه را در زیر توضیح می‌دهیم.

در این مولفه یک RectTransform که در آن همان تصویر نقشه را قرار می‌دهیم، یک عدد اعشاری برای ابعاد نقشه، یک عدد اعشاری برای انحراف دوربین برای آنکه دوربین ۹۰ درجه به سمت پایین نیست و برای نشان دادن محل انتخاب شده دوربین باید ۲۰ متر دورتر از آن قرار گیرد.

در متد Update محل دوربین را از مولفه RTSPlayer می‌گیریم.

در دو متد OnPointerDown و OnDrag که اولی زمانی که موس روی نقشه کلیک شده و دومی زمانی که موس را روی نقشه جابجا می‌کنیم است که متد MoveCamera که دوربین را تکان می‌دهد صدا می‌زنیم.

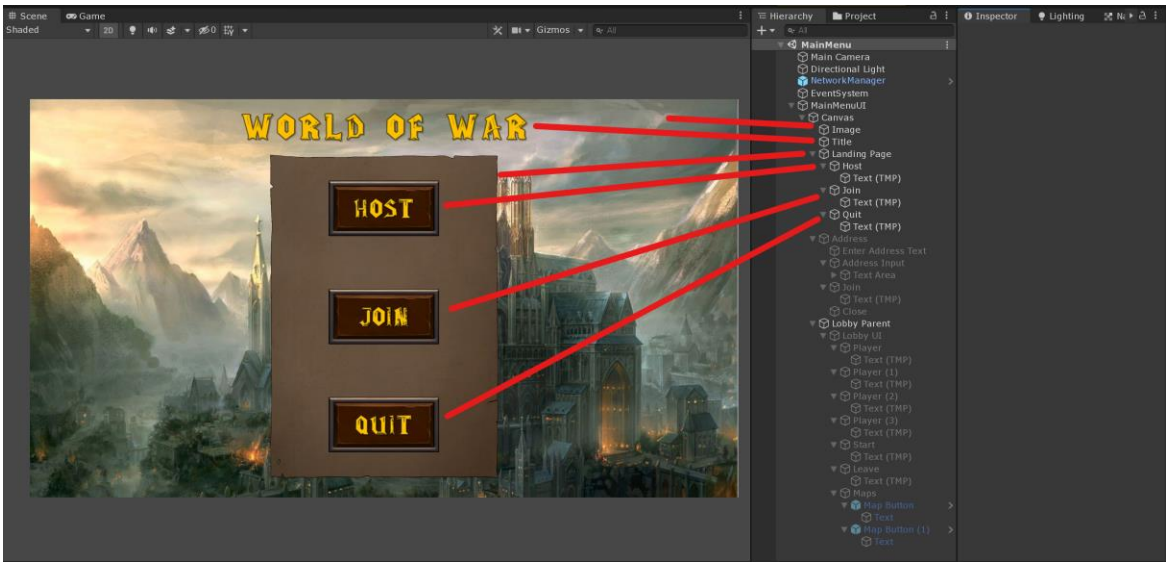
در متد MoveCamera مکان موس را می‌خوانیم سپس اگر مکان موس در چارچوب نقشه بود مکان آن را خوانده و دوربین را به آن مکان انتقال می‌دهیم.

در زیر کد این مولفه را مشاهده می‌کنیم.

```
public class Minimap : MonoBehaviour, IPointerDownHandler, IDragHandler
{
   [SerializeField] private RectTransform minimapRect = null;
   [SerializeField] private float mapScale = 20f;
   [SerializeField] private float offset = -6f;
   private Transform playerCameraTransform;
   private void Update()
   {
      if (playerCameraTransform != null) { return; }
      if (NetworkClient.connection.identity == null) { return; }
      playerCameraTransform = NetworkClient.connection.identity
         .GetComponent<RTSPlayer>().GetCameraTransform();
   }
   public void OnPointerDown(PointerEventData eventData)
   {
      MoveCamera();
   }
   public void OnDrag(PointerEventData eventData)
   {
      MoveCamera();
   }
   private void MoveCamera()
   {
      Vector2 mousePos = Mouse.current.position.ReadValue();
      if (!RectTransformUtility.ScreenPointToLocalPointInRectangle(
         minimapRect,
         mousePos,
         null,
         out Vector2 localPoint
      )) { return; }
      Vector2 lerp = new Vector2(
         (localPoint.x - minimapRect.rect.x) / minimapRect.rect.width,
         (localPoint.y - minimapRect.rect.y) / minimapRect.rect.height);
      Vector3 newCameraPos = new Vector3(
         Mathf.Lerp(-mapScale, mapScale, lerp.x),
         playerCameraTransform.position.y,
         Mathf.Lerp(-mapScale, mapScale, lerp.y));
      playerCameraTransform.position = newCameraPos + new Vector3(0f, 0f, offset);
   }
}
```

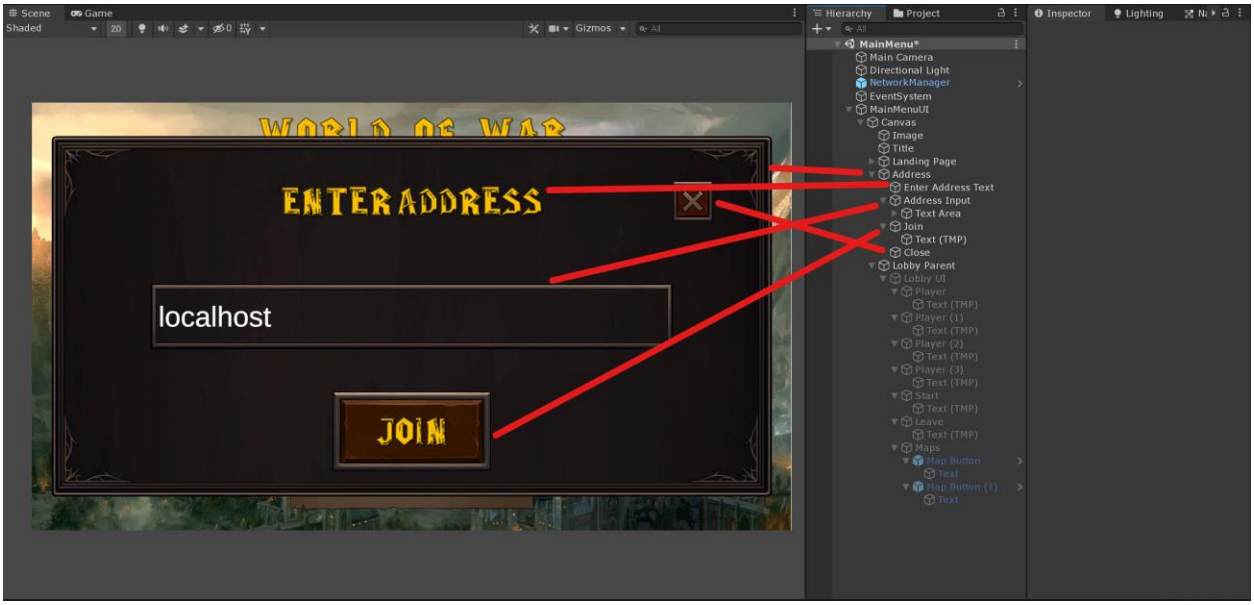
حال به ساخت صفحه ورود و صفحه لابی رسیده‌ایم که بازیکن زمان خود را از ابتدا باز کردن برنامه تا زمان شروع مسابقه را در آن صرف می‌کند. ابتدا طراحی کلی آن را در زیر می‌بینیم.

ابتدا صفحه اولیه پس از ورود را در زیر می‌بینیم که در آن دکمه Host که برای بازیکنی که می‌خواهد سرور شود، دکمه Join که برای بازیکنی که می‌خواهد به یک سرور متصل شود، دکمه Quit که دکمه خروج از بازی است، وجود دارد.



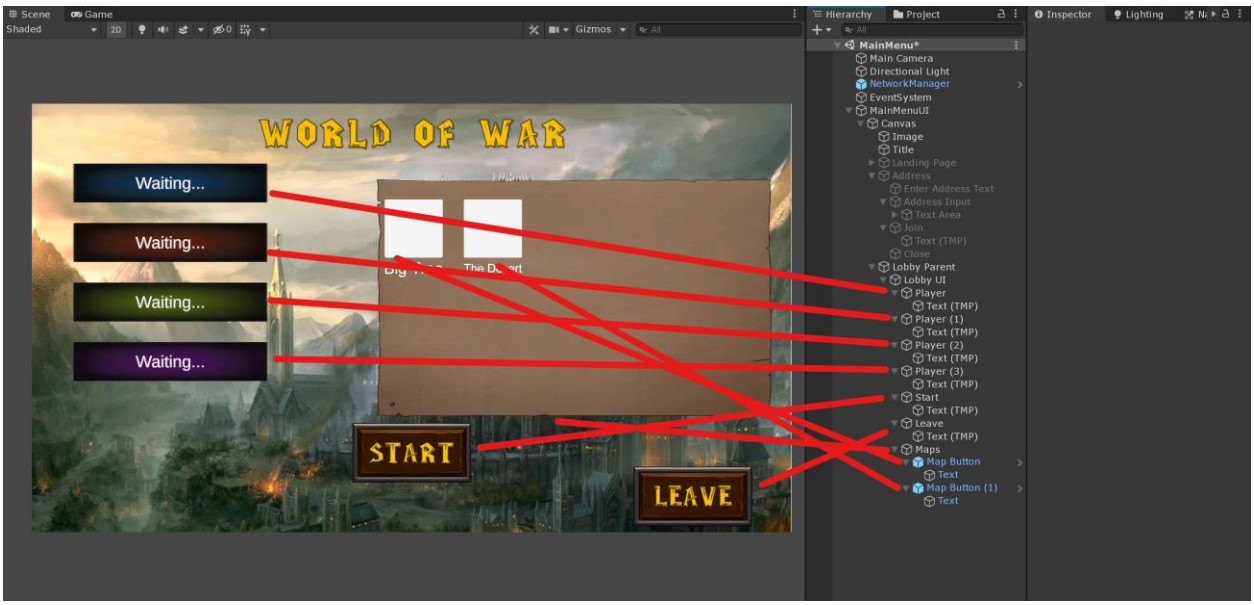
شکل ۳۰) صفحه ورود بازی

دومین صفحه، صفحه آدرس است که پس از زدن Join بازیکن به این صفحه آورده می‌شود که در آن آدرس بازیکنی که سرور است را وارد می‌کند.



شکل ۳۱) صفحه ورود آدرس بازی

بازیکن Server پس از زدن دکمه Host به و بازیکن Client پس از زدن Join در صفحه دوم به صفحه لابی آورده می‌شوند که نام آنان در سمت چپ نوشته می‌شود، هر دو بازیکن می‌توانند از بازیکن به کمک دکمه Leave خروج کنند ولی تنها بازیکن Server می‌تواند بازی را در زمان دلخواه شروع کند. انتخاب نقشه بازی نیز به عهده بازیکن Server است. این صفحه را در زیر می‌بینیم.



شکل ۳۲) صفحه لابی بازی

حال که کلیات صفحات اولیه بازی را مشاهده کردیم به تحلیل مولفه‌های نیاز برای درست کردن این سیستم لابی می‌پردازیم. برای این کار ابتدا در کد مدیر شبکه تغییرات زیر را اعمال می‌کنیم. ابتدا دو اکشن ClientOnConnected و ClientOnDisconnected را اضافه می‌کنیم تا در مولفه‌های دیگر از آن استفاده کنیم. سپس دو متد OnClientConnect و OnClientDisconnect که هنگام متصل شدن و قطع اتصال بازیکن صدا زده می‌شوند را اضافه کرده و در آن دو اکشن بالا را اضافه می‌کنیم. سپس قسمت ساختن مرکز فرماندهی را پس از اضافه شدن بازیکن پاک می‌کنیم زیرا از این پس بازیکن پس از متصل شدن بلافاصله به بازی منتقل نشده و به لابی می‌رود. تغییرات کد این مولفه را در زیر مشاهده می‌کنیم.

```
public class RTSNetworkManager : NetworkManager
{
   [SerializeField] private GameObject unitSpawnerPrefab = null;
   [SerializeField] private GameOverHandler gameOverHandlerPrefab = null;
   public static event Action ClientOnConnected;
   public static event Action ClientOnDisconnected;
   public override void OnClientConnect(NetworkConnection conn)
```

```
    {
        base.OnClientConnect(conn);
        ClientOnConnected?.Invoke();
    }
    public override void OnClientDisconnect(NetworkConnection conn)
    {
        base.OnClientDisconnect(conn);
        ClientOnDisconnected?.Invoke();
    }
    public override void OnServerAddPlayer(NetworkConnection conn)
    {
        base.OnServerAddPlayer(conn);
        RTSPlayer player = conn.identity.GetComponent<RTSPlayer>();
        player.SetTeamColor(new Color(
            UnityEngine.Random.Range(0f, 1f),
            UnityEngine.Random.Range(0f, 1f),
            UnityEngine.Random.Range(0f, 1f)
        ));
    }
}
```

حال یک مولفه برای صفحه ورودی ساخته ساخته و آن را در شی MainMenuUI قرار می‌دهیم. در کد این مولفه یک متد برای دکمه Host به اسم HostLobby قرار می‌دهیم که در آن صفحه ورودی خاموش و از مدیر شبکه StartHost را صدا می‌زنیم.

کد این مولفه را در زیر می‌بینیم.

```
public class MainMenu : MonoBehaviour
{
    [SerializeField] private GameObject landingPagePanel = null;
    public void HostLobby()
    {
        landingPagePanel.SetActive(false);
        NetworkManager.singleton.StartHost();
    }
}
```

حال مولفه‌ای به نام JoinLobbyMenu را ساخته که با استفاده از آن می‌خواهیم بازیکن Client را به لابی منتقل کنیم. در کد این مولفه به دو اکشن ClientOnConnected و ClientOnDisconnected که در RTSNetworkManager اضافه کرده‌ایم دو متد HandleClientConnected و HandleClientDisconnected را اضافه می‌کنیم. در متد HandleClientConnected که بازیکن به لابی منتقل می‌شود دکمه Join در صفحه وارد کردن آدرس فعال شده و سپس دو صفحه ابتدایی و وارد کردن آدرس را غیر فعال می‌کنیم. در متد HandleClientDisconnected نیز که بازیکن اتصال آن قطع شده دکمه Join فعال می‌شود. برای دکمه Join نیز

یک متد قرار داده‌ایم که با زدن آن آدرس از متن داخل صفحه خوانده شده و آدرس به مدیر شبکه داده شده تا بازیکن را متصل کند، پس از آن دکمه Join غیر فعال شده تا بازیکن متصل شود. کد این مولفه را در زیر می‌بینیم.

```
public class JoinLobbyMenu : MonoBehaviour
{
   [SerializeField] private GameObject landingPagePanel = null;
   [SerializeField] private TMP_InputField addressInput = null;
   [SerializeField] private Button joinButton = null;
   private void OnEnable()
   {
      RTSNetworkManager.ClientOnConnected += HandleClientConnected;
      RTSNetworkManager.ClientOnDisconnected += HandleClientDisconnected;
   }
   private void OnDisable()
   {
      RTSNetworkManager.ClientOnConnected -= HandleClientConnected;
      RTSNetworkManager.ClientOnDisconnected -= HandleClientDisconnected;
   }
   public void Join()
   {
      string address = addressInput.text;
      NetworkManager.singleton.networkAddress = address;
      NetworkManager.singleton.StartClient();
      joinButton.interactable = false;
   }
   private void HandleClientConnected()
   {
      joinButton.interactable = true;
      gameObject.SetActive(false);
      landingPagePanel.SetActive(false);
   }
   private void HandleClientDisconnected()
   {
      joinButton.interactable = true;
   }
}
```

حال یک مولفه برای صفحه لابی ساخته که در آن باید دکمه خارج شدن از لابی را کدنویسی کنیم. برای این کار یک متد به نام LeaveLobby ساخته که در آن اگر بازیکن Server بود سرور را بسته و همه بازیکنان به صفحه ابتدایی انتقال داده می‌شوند و اگر بازیکن Client بود اتصال آن به سرور قطع شده و به صفحه اولیه انتقال داده می‌شود. همچنین برای فهمیدن متصل شدن، HandleClientConnected که لابی را روشن کرده به اکشن ClientOnConnected که در مدیر شبکه موجود است اضافه می‌کنیم.
کد این مولفه را در زیر می‌بینیم.

```
public class LobbyMenu : MonoBehaviour
```

```
{
  [SerializeField] private GameObject lobbyUI = null;
  private void Start()
  {
    RTSNetworkManager.ClientOnConnected += HandleClientConnected;
  }
  private void OnDestroy()
  {
    RTSNetworkManager.ClientOnConnected -= HandleClientConnected;
  }
  private void HandleClientConnected()
  {
    lobbyUI.SetActive(true);
  }
  public void LeaveLobby()
  {
    if (NetworkServer.active && NetworkClient.isConnected)
    {
      NetworkManager.singleton.StopHost();
    }
    else
    {
      NetworkManager.singleton.StopClient();
      SceneManager.LoadScene(0);
    }
  }
}
```

حال برای آنکه سازنده بازی، بازی را از لابی شروع کند نیاز به تغییرات زیر داریم.

ابتدا در کد مدیر شبکه یک بولین به اسم isGameInProgress می‌سازیم که زمانی که بازی شروع شد true می‌شود، سپس یک لیست از تمام بازیکنان درست کرده تا بازیکنان را در یک لیست داشته باشیم. سپس متد OnServerConnect که در آن اگر بازیکنی در حین بازی موفق به اتصال به سرور شد آن را قطع کند می‌سازیم، پس از آن متد OnServerDisconnect که در آن بازیکن را از لیست بازیکنان سرور پاک می‌کنیم. در زمان خاموش شدن(OnStopServer) سرور لیست بازیکنان را خالی می‌کنیم. یک متد برای شروع بازی می‌سازیم که تنها زمانی بازی را شروع کند که تعداد بازیکنان دو یا بیشتر باشد. در متد OnServerAddPlayer بازیکن را به لیست بازیکنان اضافه می‌کنیم و اگر بازیکن اولین نفر بود که وارد لیست شده به این معنی است که Server است و آن بازیکن را PartyOwner می‌کنیم. کد این مولفه را به صورت کامل در زیر می‌بینیم.

```
public class RTSNetworkManager : NetworkManager
{
  [SerializeField] private GameObject unitBasePrefab = null;
  [SerializeField] private GameOverHandler gameOverHandlerPrefab = null;
  public static event Action ClientOnConnected;
  public static event Action ClientOnDisconnected;
  private bool isGameInProgress = false;
  public List<RTSPlayer> Players { get; } = new List<RTSPlayer>();
  #region Server
```

```
public override void OnServerConnect(NetworkConnection conn)
{
    if (!isGameInProgress) { return; }
    conn.Disconnect();
}
public override void OnServerDisconnect(NetworkConnection conn)
{
    RTSPlayer player = conn.identity.GetComponent<RTSPlayer>();
    Players.Remove(player);
    base.OnServerDisconnect(conn);
}
public override void OnStopServer()
{
    Players.Clear();
    isGameInProgress = false;
}
public void StartGame()
{
    if (Players.Count < 2) { return; }
    isGameInProgress = true;
    ServerChangeScene("Scene_Map_01");
}
public override void OnServerAddPlayer(NetworkConnection conn)
{
    base.OnServerAddPlayer(conn);
    RTSPlayer player = conn.identity.GetComponent<RTSPlayer>();
    Players.Add(player);
    player.SetTeamColor(new Color(
        UnityEngine.Random.Range(0f, 1f),
        UnityEngine.Random.Range(0f, 1f),
        UnityEngine.Random.Range(0f, 1f)
    ));
    player.SetPartyOwner(Players.Count == 1);
}
public override void OnServerSceneChanged(string sceneName)
{
    if (SceneManager.GetActiveScene().name.StartsWith("Scene_Map"))
    {
        GameOverHandler gameOverHandlerInstance = Instantiate(gameOverHandlerPrefab);
        NetworkServer.Spawn(gameOverHandlerInstance.gameObject);
        foreach(RTSPlayer player in Players)
        {
            GameObject baseInstance = Instantiate(
                unitBasePrefab,
                GetStartPosition().position,
                Quaternion.identity);
            NetworkServer.Spawn(baseInstance, player.connectionToClient);
        }
    }
}
#endregion
#region Client
public override void OnClientConnect(NetworkConnection conn)
{
    base.OnClientConnect(conn);
    ClientOnConnected?.Invoke();
```

```
    }
    public override void OnClientDisconnect(NetworkConnection conn)
    {
        base.OnClientDisconnect(conn);

        ClientOnDisconnected?.Invoke();
    }
    public override void OnStopClient()
    {
        Players.Clear();
    }
    #endregion
}
```

حال در کد مولفه LobbyMenu نیز تغییراتی ایجاد می‌کنیم که در زیر توضیح داده‌ایم. یک متد به اسم StartGame برای قرار دادن در دکمه شروع کردن بازی ساخته و در آن CmdStartGame را از مولفه بازیکن صدا می‌زنیم که در این Command تنها بازیکنی می‌تواند بازی را شروع کند که PartyOwner باشد. کد اضافه شده را در زیر می‌بینیم.

```
    public void StartGame()
    {
        NetworkClient.connection.identity.GetComponent<RTSPlayer>().CmdStartGame();
    }
```

تا به این لحظه بازی بر روی حالت Local بازی می‌شد حال برای آنکه بازی را به شبکه اینترنت متصل کرده و بتوانیم آن را روی Steam که یک پلتفورم بازی جهانی است بازی کنیم، ابتدا پکیج FizzySteamWorks را به پروژه اضافه می‌کنیم سپس تغییرات زیر را در مولفه MainMenu اعمال می‌کنیم.

ابتدا یک بولین برای آنکه بتوانیم بین دو حالت Local و Steam جابه‌جا شویم اضافه می‌کنیم، سپس سه Callback برای ساخته شدن لابی، درخواست اضافه شدن به لابی و ورود به لابی درست می‌کنیم. در متد Start اگر در حالت استیم بود سه Callback را می‌سازیم. در متد HostLobby اگر در حال استفاده از استیم بود لابی را می‌سازیم. در متد OnLobbyCreated اگر ساخت لابی موفق بود صفحه اصلی را روشن می‌کنیم، سپس توسط مدیر شبکه بازیکن Host شده و اسم لابی در پلتفورم استیم به اسم بازیکن سازنده در می‌آید.
در متد OnGameLobbyJoinRequested درخواست اضافه شدن به بازی را توسط بازیکن دیگر ثبت می‌کنیم.
در متد OnLobbyEntered به سرور ملحق شده و صفحه اصلی را خاموش می‌کنیم.
در زیر کد کامل این مولفه را مشاهده می‌کنیم.

```
public class MainMenu : MonoBehaviour
{
    [SerializeField] private GameObject landingPagePanel = null;
```

```
    [SerializeField] private bool useSteam = false;
    protected Callback<LobbyCreated_t> lobbyCreated;
    protected Callback<GameLobbyJoinRequested_t> gameLobbyJoinRequested;
    protected Callback<LobbyEnter_t> lobbyEntered;
    private void Start()
    {
        if (!useSteam) { return; }
        lobbyCreated = Callback<LobbyCreated_t>.Create(OnLobbyCreated);
        gameLobbyJoinRequested = Callback<GameLobbyJoinRequested_t>.Create(OnGameLobbyJoinRequested);
        lobbyEntered = Callback<LobbyEnter_t>.Create(OnLobbyEntered);
    }
    public void HostLobby()
    {
        landingPagePanel.SetActive(false);
        if (useSteam)
        {
            SteamMatchmaking.CreateLobby(ELobbyType.k_ELobbyTypeFriendsOnly, 4);
            return;
        }
        NetworkManager.singleton.StartHost();
    }
    private void OnLobbyCreated(LobbyCreated_t callback)
    {
        if (callback.m_eResult != EResult.k_EResultOK)
        {
            landingPagePanel.SetActive(true);
            return;
        }
        NetworkManager.singleton.StartHost();
        SteamMatchmaking.SetLobbyData(
            new CSteamID(callback.m_ulSteamIDLobby),
            "HostAddress",
            SteamUser.GetSteamID().ToString());
    }
    private void OnGameLobbyJoinRequested(GameLobbyJoinRequested_t callback)
    {
        SteamMatchmaking.JoinLobby(callback.m_steamIDLobby);
    }
    private void OnLobbyEntered(LobbyEnter_t callback)
    {
        if (NetworkServer.active) { return; }
        string hostAddress = SteamMatchmaking.GetLobbyData(
            new CSteamID(callback.m_ulSteamIDLobby),
            "HostAddress");
        NetworkManager.singleton.networkAddress = hostAddress;
        NetworkManager.singleton.StartClient();
        landingPagePanel.SetActive(false);
    }
}
```

فصل چهارم

مقایسه پروژه با نمونه‌های دیگر

٤-١- مقدمه

در این فصل هر یک از نمونه‌هایی که در فصل دوم نام برده و توضیح داده‌ایم را با پروژه توسعه داده شده مقایسه می‌کنیم. در این مقایسه ابتدا ویژگی‌های نمونه‌ها را نام برده و توضیح می‌دهیم و سپس وجود یا عدم وجود این ویژگی را در پروژه بررسی می‌کنیم. سپس نقاط قوت و ضعف پروژه را نسبت به نمونه‌ها بیان می‌کنیم.

٤-٢- مقایسه

در ابتدا به توضیح کوتاهی از نمونه‌های بازی‌های استراتژی زمان واقعی و ویژگی‌های آنان می‌پردازیم.

٤-٢-١- Warcraft III

بازی فانتزی و استراتژی زمان واقعی است که توسط Entertainment Blizzard و در ژوئیه ۲۰۰۲ منتشر شد. در این بازی ، مانند بسیاری از بازی‌های استراتژی زمان‌واقعی ، بازیکنان منابع را جمع‌آوری می‌کنند ، واحدها و قهرمانان را آموزش می‌دهند و پایگاه‌هایی را برای دستیابی به اهداف مختلف در حالت تک نفره یا شکست دادن بازیکن آنلاین رقیب ایجاد میکنند. چهار جناح قابل بازی را می توان انتخاب کرد: انسان ها، اورک ها الف ها و نامیرایان. کمپین تک نفره وارکرافت به صورت پیشرونده بیان می‌شود که شامل داستان هر چهار جناح می‌باشد.

یکی از ویژگی‌های این بازی کمپین منحصر به فرد آن که دارای قدرت داستان گویی فوق‌العاده است می‌باشد؛ این داستان بسیار پهناور می‌باشد که پس از ساخت بازی در چندین کتاب تمام داستان بیان می‌شود. یکی دیگر از ویژگی‌های آن داشتن یک یا چند شخصیت قهرمان برای هر بازیکن که قدرت‌های مخصوص خود را دارند می‌باشد؛ بازیکن زمانی که قهرمان را انتخاب میکند میتواند به کمک رابط کاربری و یا دکمه‌های کیبورد QWER این قدرت‌ها را استفاده کند مانند اضافه کردن سلامتی یه نیرو خودی یا به وجود آوردن یک خرس برای کمک به بازیکن و قدرت‌های بسیار زیاد دیگر. یکی از مشکلات آن سرور خصوصی آن است که به دلیل آنکه شرکت در ساخت بازی از سرور خصوصی به جای ساخت سیستم Client-Server که در آن یکی از بازیکنان سرور باشد است و دیگر بازیکنان به آن سرور متصل شوند، همه بازیکنان مجبور به متصل شدن به سرور خریداری شده توسط شرکت متصل می‌شوند زیرا پس از چند سال از انتشار بازی زمانی که سرور آن تعطیل شد، باعث بازنشستگی بازی شد. یکی دیگر از مشکلات آن ساخت این بازی به وسیله موتور خود شرکت است که

باعث شده است بازسازی این بازی پس از ۱۵ سال از ساخت نسخه اصلی آن بسیار مشکل شده و نسخه بازسازی آن دارای اروورها فراوان باشد که طبق رای مردم و منتقدین این بازسازی جزو بدترین بازی‌های ساخته شده محسوب شود.

در پروژه ما کمبود بخش کمپین که دارای داستان منحصر به فرد و بازی تک نفره با هوش مصنوعی قوی باشد به چشم می‌خورد که در بازی وارکرفت این نکته لحاظ شده است. همچنین عدم وجود جناح‌های مختلف که بازیکنان را جذب کند تا به رقابت با هم بپردازند نیز در پروژه ما دیده می‌شود. در شکل ۳۳ تصویری از جناح‌های مختلف می‌بینیم. برتری پروژه ما نسبت به وارکرفت این است که سیستم Client-Server است که باعث می‌شود نیازی به سرور خصوصی نداشته باشیم و هزینه پروژه را به شدت کاهش می‌دهد. همچنین ساخت پروژه ما به وسیله موتور یونیتی باعث می‌شود که پروژه به آرامی پس از مدت‌ها غیرقابل بازسازی نشود.

شکل ۳۳ : جناح‌های مختلف وارکرفت

Age of Empires II -۲-۲-٤

بازی استراتژی زمان واقعی است که توسط Studios Ensemble توسعه یافته و توسط مایکروسافت منتشر شده است. این بازی در سال ۱۹۹۹ برای مایکروسافت ویندوز و مکینتاش منتشر شد. عصر پادشاهان در قرون وسطی تنظیم شده و شامل سیزده تمدن قابل بازی است. هدف بازیکنان جمع آوری منابع است که از آنها برای ساختن شهر ، ایجاد ارتش و شکست دشمنان خود استفاده می‌کند. پنج کمپین تاریخی وجود دارد که بازیکن را به شرایط تخصصی و پشتیبانی از داستان و همچنین سه حالت بازی تک نفره اضافی محدود می‌کند. همچنین حالت آنلاین چند نفره نیز پشتیبانی می شود.

انواع گوناگون منابع مانند غذا ، چوب ، سنگ و طلا و انواع گوناگون تمدن‌ها ، انبارهای منابع ، کمپین و نیروها باعث می‌شود این بازی گیم‌پلی وسیعی داشته باشد و به سادگی بازیکن از آن خسته نشود و تا مدت‌ها به تجربه‌های جدید دست یابند. در شکل ۳٤ تصویری از مزارع بازی مشاهده می‌کنید. یکی دیگر از ویژگی‌های آن استفاده از پلتفرم Steam برای بخش آنلاین است که باعث می‌شود این بازی به سادگی توسط بازیکنان در دسترس بوده و رقابت آنلاین به سادگی در دسترس بوده و پیدا کردن رقیبان به امکان‌پذیر باشد.

یکی از ضعف‌های این بازی رابط کاربری آن است که به نسبت بازی‌های دیگر باری کاربرانی که آسنایی قبلی ندارند پیچیده است و دارای ظاهر زیبایی نیست. همچنین به دلیل استفاده از موتور و کد نسخه قبلی این بازی دارای هوش مصنوعی ضعیف و وجود باگ‌های فراوان و کرش کردن برنامه است و باعث طولانی شدن پروسه ساخت بازی به دلیل استفاده از موتور قدیمی است.

در پروژه ما به دلیل نبود انواع زیاد نیرو و منابع و نبود حالت آفلاین و کمپین ممکن است کاربر زودتر از حد معمول خسته شود. در این بازی مانند پروژه ما به دلیل استفاده از Steam حالت آنلاین به سادگی در دسترس است. برتری پروژه ما نسبت به این بازی رابط کاربری آسان آن است که باعث می‌شود بازی در دسترس کاربران بیشتری باشد و نیاز به یادگیری زیادی در آن نباشد. همچنین به دلیل استفاده از موتور یونیتی هوش مصنوعی قوی‌تری در پروژه ما وجود دارد و کدهای آن پیچیدگی زیادی ندارند و پروسه ساخت را کوتاه‌تر می‌کند.

شکل ۳٤ : مزارع و روستایان بازی age of empires

Stronghold: Crusader –۳–۲–٤

بازی ساخت Firefly Studios که در طول جنگ‌های صلیبی در خاورمیانه بازی می‌شود. جانشین بازی استراتژیک Stronghold ۲۰۰۱ است. بازی Crusader شباهت‌های زیادی با Stronghold اصلی دارد ، اما با این نسخه قبلی تفاوت دارد زیرا بازی دیگر در انگلستان بازی نمیشود ، یکی دیگر از ویژگی‌های برجسته که در نسخه قبلی آن یافت نشد ، حالت تقابل تک نفره است که به جای مبارزات خطی امکان نبردهای پیچیده‌تر را با رقیبان هوش مصنوعی را فراهم می‌کند.

یکی از ویژگی‌های این بازی این است که در آن مزارع تنها روی چمن‌هایی که به دلیل محیط خاورمیانه‌ای آن کم است و همچنین معادن آن که محدود بوده تنها در مکان‌هایی که سنگ یا آهن وجود دارند ساخته می‌شود که باعث رقابت بیشتر بازیکنان برای به دست آوردن منابع می‌شود و بازی و رقابت را جذاب‌تر می‌کند. یکی دیگر از ویژگی‌های آن که بازی را سخت‌تر کرده آن را خاص‌تر و جذاب‌تر نیز می‌کند این است که در آن تولید رعیت‌ها و مردم عادی داخل پادشاهی هر بازیکن وابسته به شاد بودن آن‌ها کمتر می‌شود و درنتیجه سربازان کمتری تولید می‌شود و این کار باعث می‌شود که بازیکن علاوه بر رویاروی با دشمن و جمع کردن منابع چالش جدیدی برای بازیکن که باید برای راضی نگه داشتن مردم خانه‌ها و مسجدها و پارک‌هایی نیز بسازد و خاصیت یک بازی شهرسازی را به این شکل داشته باشد. در شکل ۳۵ خانه ها و چاه های ساخته شده برای مردم را مشاهده می‌کنیم. یکی از ضعف‌های این بازی کند شدن و افت فریم بازی پس از ساختن تعداد بالای سرباز و نیرو در آن است که رهبری کردن و حرکت دادن نیروها را بسیار مشکل می‌کند. یکی دیگر از مشکلات آن بخش کمپین و افلاین آن است که بسیار کوتاه بوده و پس از تنها چند ساعت بازی به پایان می‌رسد و داستان خاص و ویژه‌ای را دنبال نمی‌کند در حالی که می‌توانست داستان غنی را به دلیل وجود جنگ‌های صلیبی در این دوران دنبال کند. همچنین هوش مصنوعی ضعیف رقیبان بخش آفلاین آن نیز باعث ضعیف‌تر شدن بخش آفلاین می‌شود و باعث می‌شود بازیکنان از رقابت با هوش مصنوعی بازی لذت زیادی نبرند.

پروژه ما به دلیل عدم وجود مردم عادی در پادشاهی بازیکن باعث ضعیف شدن جذابیت بازی می‌شود و باعث می‌شود تنها چالش بازی بازیکنان رقیب و جمع‌آوری منابع باشد. همچنین نبود مکان‌های خاص برای استخراج طلا باعث می‌شود بازیکنان برای به دست آوردن منابع چالش بسیار زیادی پیش روی خود نداشته باشند.

در پروژه ما با افزایش تعداد نیروها بازی به دلیل وجود پیش‌ساخته‌ها در صحنه بازی مشکل کند شدن را ندارد و بازیکن با بزرگ تر شدن ارتش خود و رقیبان مشکل افت فریم و درنتیجه آن حرکت و رهبری را نخواهد داشت.

شکل ۳۵ : خانه‌های بازی stronghold crusader

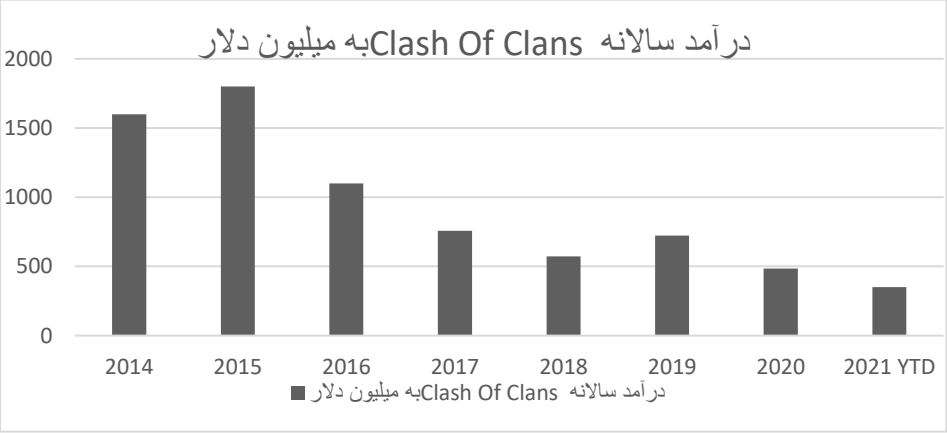
Clash of Clans –٤–٢–٤

یک بازی استراتژی موبایل رایگان با پرداخت درون برنامه‌ای است که توسط توسعه دهنده فنلاندی Supercell توسعه و منتشر شده است. بازی در دنیایی با تم فانتزی تنظیم میشود که در آن بازیکن رئیس یک روستا است. کلش‌آف‌کلنز بازیکنان را مجبور می کند تا با استفاده از منابعی که از حمله به روستاهای بازیکنان دیگر به دست آورده‌اند ، دهکده خود را بسازند. کسب پاداش ، خرید آن‌ها با مدال یا تولید آن‌ها در روستای خود. برای حمله ، بازیکنان انواع مختلفی از نیروها را با استفاده از منابع آموزش می دهند. منابع اصلی طلا ، اکسیر و اکسیر سیاه هستند. بازیکنان میتوانند برای ایجاد قبیله ، گروه‌هایی تا پنجاه نفر به هم متصل شوند ، که میتوانند با هم در جنگ‌های قبیله‌ای شرکت کنند ، نیرو اهدا کنند و نیرو دریافت کنند و با یکدیگر چت کنند.

یکی از ویژگی‌های آن که آن را بسیار محبوب کرده است قابلیت ساخت کلن به همراه دوستان و حمله و جنگ‌های بین کلن‌ها در آن است که باعث می‌شود بازیکنان دوستان خود را به بازی دعوت کنند و جمع کلن خود و در نتیجه کاربران بازی را افزایش دهند. یکی دیگر از دلایل وجود کاربرانی که همیشه به بازی برمیگردند این است که رشد پادشاهی یک بازیکن به آمدن چند ساعت و چند روز یک بار بازیکن و ساختن و پیشرفت قلعه آن است که حس پیشرفت و ترقی را به بازیکن منتقل می‌کند. یکی دیگر از ویژگی‌های بازی که باعث درآمد

بالای آن سات این است که با وجود رایگان بودن نصب بازی قابلیت خرید درون برنامه‌ای آن است که به کمک آن بازیکن پیشرفت‌های خود را با خرید پول داخل برنامه سرعت می‌بخشد و باعث شده است در ۸ سالی که از ساخت آن گذشته است بیش از ۷ میلیارد دلار درآمد داشته باشد. در نمودار ۱۵ درآمد سالانه این بازی را مشاهده می‌کنیم.

یکی از ضعف‌های بازی نبود حالت آفلاین و هوش مصنوعی برای مقابله با بازیکن است که در آن بتواند بازیکن به هوش مصنوعی‌هایی حمله و روی آن‌ها تمرین کرده و آماده رویاروی با بازیکنان آنلاین بشود، همچنین نبود بخش داستانی نیز برقرار کردن ارتباط بازیکن با بازی و شخصیت‌های آن را سخت می‌کند. وجود پرداخت درون برنامه‌ای در بازی نیز همچنان که یک ویژگی و باعث درآمد بالای آن است یکی از ضعف‌های آن نیز می‌باشد زیرا باعث می‌شود بازیکنان با پرداخت‌های درون برنامه‌ای از بازیکنانی که تنها در بازی وقت گذاشته‌اند جلو افتاده و باعث ناامید شدن بازیکنان شده و آن‌ها را از بازی دور کند و باعث کاهش کاربران بازی شود.



نمودار ۱۵ : درآمد سالیان بازی کلش‌آف‌کلنز به میلیون دلار در ۸ سال گذشته

Northgard –۵–۲–٤

با الهام از اساطیر اسکاندیناوی یک بازی استراتژیک زمان واقعی است که در آن یک قبیله وایکینگ‌ها سعی می‌کنند کنترل یک قاره وحشی را به دست بگیرند. در این بازی چندین قبیله وایکینگ با کشتی به یک منطقه وارد شده و شروع به ساخت ساختمان‌های خود می‌کنند، نیروهای آن تنوع زیادی دارند و بیشتر آنان از نوع غیر سرباز هستند مانند: شکارچی ، کشاورز ، شفادهنده ، معدن‌چی ، دریانورد و برده می‌باشد. در این بازی

تمرکز بیشتر بر روی ساخت یک جامعه که قسط زنده ماندن در زمستان‌های طاقت‌فرسای بازی را دارد می‌باشد و نیروهای کمتری به عنوان سرباز وجود دارد.

یکی از ویژگی‌های این بازی جدا کردن مناطق و محدود کردن ساخت در هر منطقه است ، به کمک این ویژگی وقتی بازیکن بازی را شروع می‌کند تنها بخش خود را شناخته و به دلیل رندوم بودن شکل مناطق نمی‌داند که دیگر مناطق تحت کنترل چه موجوداتی است و باید با کشف هر سرزمین توسط نیروی پیش‌آهنگ آن را کشف و سپس توسط سربازان به آن حمله کرده و تحت کنترل خود درآورد. به دلیل رندوم بودن ساخت نقشه توسط بازی هر بار که بازیکنان وارد یک مسابقه جدید می‌شوند نقشه‌ای جدید را مشاهده می‌کنند و این کار باعث شادابی و به وجود آمدن نقشه‌های متفاوت در هر بازی می‌شود. یکی دیگر از ویژگی‌های آن انواع محتلف قبیله‌ها می‌باشد که باعث می‌شود بازیکنان انتخابات زیادی داشته باشند برای مثال یک قبیله بر روی دریانوردی تمرکز بیشتری دارد و دیگری به برده‌داری شهرت دارد.

یکی از مشکلات بزرگ آن نبود ساختمان‌های دفاعی و سربازان متنوع است که باعث می‌شود بخش‌های جنگی آن خسته کننده باشد و تمرکز بازیکنان بیشتر بر روی زنده ماندن از زمستان‌ها و نیروهای وحشی باشد. یکی دیگر از مشکلات آن هوش مصنوعی رقیبان آفلاین آن است که بسیار ضعیف بوده و باعث می‌شود بازیکن در بخش آفلاین بدون مشکلی هوش مصنوعی آن را شکست داده و چالشی در آن نداشته باشد. از دیگر مشکلات بازی می‌توان به عدم وجود قابلیت گروه‌سازی نام برد که درنتیجه تمام بازیکنان با یکدیگر رقیب هستند.

پروژه ما مشکل نداشتن سربازان متنوع را ندارد و بازیکنان انتخاب‌های زیادی در ساخت سربازان دارند ولی نبود محدود کننده ساخت و جدا کردن مناطق به چشم می‌خورد که باعث می‌شود بازیکنان چالش‌های کمتری در بازی داشته باشند که از جذابیت بازی کم می‌کند.

در شکل ۳٦ مناطق مختلف اطراف بازیکن را مشاهده می‌کنیم و خطوط زرد که نشان دهنده امکان حرکت بین مناطق و خطوط قرمز نشان دهنده عدم امکان حرکت به دلیل وجود دیوار سنگی یا دریاچه را می‌بینم. در نتیجه بازیکن از یک منظر دسترسی کمتری نسبت به محیط اطراف دارد و از منظری دیگر در مقابل دشمنان دیواری طبیعی و امنیت نسبی بیشتری دارد.

شکل ۳٦ : مناطق و امکان حرکت بین آنان در بازی Northgard

٤-۲-٦- Crusader Kings III

یک بازی استراتژیک بزرگ و شبیه ساز سلسله است که در قرون وسطی اتفاق می افتد. بازیکنان می‌توانند تاریخ شروع ۸٦۷ یا ۱۰٦٦ را انتخاب کنند و تا سال ۱٤٥۳ بازی کنند. سلسله‌ها می‌توانند شاخه‌هایی تشکیل دهند که سر خود را دارند و عمدتاً مستقل از سلسله والدین خود عمل می‌کنند. در این بازی بازیکنان کنترل یک سلسله که بر مناطقی از آسیا و یا اروپا را حکومت می‌کنند به دست می‌گیرد و دستورات سیاسی یا جنگی بزرگی را می‌تواند صادر کند مانند ازدواج یک فرزند با بزرگی از سلسله دیگر یا دستور حمله به یک کشور مانند انگلستان.

یکی از ویژگی‌های آن وجود شخصیت‌ها و وقایع تاریخی به صورت کامل است که بازیکن می‌تواند در آن نقش داشته باشد و به صورتی که علاقه دارد تاریخ را تغییر دهد. یکی دیگر از ویژگی‌های آن وجود انتخاب بین سیاست و جنگ است که با وجود آن می‌توانیم بدون نیاز به جنگ و ارتش یک کشور را تحت سلطه خود درآوریم. یکی دیگر از ویژگی‌های آن نقشه پهناور بازی است که باعث می‌شود بازیکنان مدت زیادی را سرگرم بازی و فتح کشورها باشند. یکی از مشکلات آن سیستم‌های حمله و دفاع آن است، از آنجایی که بیشتر تمرکز بازی بر روی سیستم‌های سلسله و کنترل خاندان‌ها است زیاد به سیستم‌های کنترل ارتش آن و نیروهای دریایی آن توجه نشده است و بازی دچار این خلا است. یکی دیگر از مشکلات بازی پیچیده بودن رابط کاربری و به طور کلی سیستم‌های بازی است که باعث می‌شود بسیاری از بازیکنان که توجه‌های زیادی به یادگیری اولیه

ندارند از بازی خسته شده و آن را کنار بگذارند و همچنین به دلیل پیچیده بودن بازی بازیکنان رده سنی پایین به آن توجه‌ای نداشته باشند.

پروژه ما به دلیل عدم وجود سیستم سلسله‌ای و کنترل خاندان بسیار ساده است و بازیکنان حرفه‌ای که نیاز به چالش‌های سخت دارند در این بازی آن را مشاهده نمی‌کنند و باعث زدگی آنان از بازی می‌شود ولی به دلیل عدم وجود آن بازیکنان با رده سنی پایین سختی در آن مشاهده نکرده و به بازی جذب می‌شوند. همچنین به دلیل کوچک بودن نقشه پروژه ما بازیکنان هر مسابقه را سریع‌تر پیش می‌برند که باعث می‌شود بازی سریع منسوخ شده و بازیکنان از آن فاصله بگیرند.

در شکل ۳۷ از سیستم خاندان و اشخاص مقام‌ها را در این بازی می‌بینیم. سمت چپ بالا شخصیت بازیکن در سمت راست آن همسر و در سمت راست همسر فرزند جانشین آن ، در سمت چپ صفحه شخصیت-های مقام‌دار بازیکن را مشاهده می‌کنیم.

شکل ۳۷ : خاندان و مقامات سلسله بازیکن در بازی Crusader Kings III

فصل پنجم

نتیجه‌گیری و پیشنهادات

۵-۱- مقدمه

در این فصل ابتدا نکات قابل بهبود که به دو بخش کلی(کاربر محور بودن، متمایز بودن، استفاده از پیش‌ساخته) و ویژه استراتژی زمان‌واقعی(بخش کمپین و داستانی، گروه‌بندی و ساخت کلن) تقسیم شده است را بیان می‌کنیم، سپس به بیان اهمیت‌ها و چالش‌ها مانند تحریم‌ها و تاثیر آن‌ها بر روی ساخت بازی و نبود حمایت از بازی‌سازان و تاثیر آن در این پروژه می‌پردازیم و در آخر به گرفتن نتایج پایانی و پیشنهادات برای آینده پروژه می‌پردازیم که در این پیشنهادات اضافه کردن مواردی همچون بخش داستانی و کمپین، جناح و تمدن‌های مختلف، وجود رعیت و مردم عادی، گروه‌بندی و ساخت کلن و پرداخت درون‌برنامه‌ای است را بررسی می‌کنیم.

۵-۲- نکات قابل بهبود کلی هر پروژه بازی

ابتدا مواردی که باید در هر پروژه بازی‌سازی رعایت شوند را در زیر ذکر کرده‌ایم و وجود یا عدم وجود آن را در پروژه خود سنجیده‌ایم.

۵-۲-۱- کاربر محور بودن

قبل از اینکه شروع به ساخت یک بازی کنیم ، باید یک ایده واضح از اینکه کاربر هدف ما کیست ایجاد کنیم. ایجاد پرسونا در مورد انواع افرادی که به بازی ما جذب می‌شوند نیز مفید است. علاوه بر این، باید تحقیقات بازار را در مورد خواسته‌ها، علاقه‌مندی‌ها و نیازهای مخاطبان هدف خود را انجام دهیم تا بتوانیم محصولی بسازیم که به این علایق پاسخ دهد و تجربه کاربری (UX) را بهبود بخشد. این نکته در پروژه ما رعایت شده است و سعی در آن داشته‌ایم تا خواسته کاربران که یک بازی استراتژی آنلاین با مضامین ایرانی و زبان فارسی است را رعایت کرده‌ایم.

۵-۲-۲- متمایز بودن

اکنون که مخاطبان خود را درک کرده‌ایم، باید راه‌هایی را نیز شناسایی کنیم تا بازی خود را از محصولات شناخته شده متمایز کنیم. بازار بازی بسیار بزرگ و بسیار شلوغ است، بنابراین دیدگاه ما برای برنامه باید چیزی را ارائه دهد که رقبا ارائه نمی‌دهند – عناصری که کاربران نمی‌توانند در هیچ جای دیگری پیدا کنند.

پروژه ما این نکته را رعایت کرده است و بسیار متفاوت‌تر از پروژه‌های موجد در بازار است.

۵-۲-۳- کدها را در یک NameSpace قرار دهیم

این از برخورد کد بین کتابخانه‌ها و کدهای شخص ثالث جلوگیری می‌کند. اما برای جلوگیری از درگیری با کلاس‌های مهم، از "Object" یا "Action" یا "Event" به عنوان نام کلاس استفاده نکنیم.

در پروژه ما تمام کدها که مربوط به یک سیستم می‌باشند دارای namespace خود می‌باشند و این کار باعث شده است وابستگی بین مولفه‌ها کاهش یابد.

۵-۲-٤- استفاده از پیش‌ساخته‌ها

استفاده از پیش‌ساخته ها باعث روان‌تر شدن عملیات ساخت نیروها و ساختمان‌ها شده است و حجم نرم‌افزار را کاهش و باعث افزایش فریم آن نسبت به حالت عادی می‌شود.

در پروژه ما این نکته رعایت شده است که در فصل پیاده سازی پروژه آن را دیدیم.

۵-۳- نکات قابل بهبود ویژه پروژه استراتژی زمان واقعی

به طور کلی چالش‌هایی که در پروژه ما وجود دارد و نبود این نکات به چشم می‌خورد و اضافه کردن آن می‌تواند باعث بهبود پروژه بشود که در پروژه‌های مثال زده شده‌ی بالا وجود دارند را در زیر می‌بینیم.

۵-۳-۱- بخش داستانی و کمپین

این بخش از هر بازی استراتژی زمان واقعی بخشی آفلاین است که در آن بازیکن با رقیبان هوش مصنوعی که بازی به صورت داستانی جذاب روایت می‌کند و باعث می‌شود بازیکن با شخصیت‌ها و گیم پلی بازی آشنا شده و در بعضی بازی‌ها روایت جنگی تاریخی را شرح می‌دهد رقابت کند. در پروژه ما این بخش به خاطر کمبود هوش مصنوعی قوی برای رقیبان آفلاین وجود ندارد و وجود آن می‌تواند به پروژه کمک شایانی بکند.

۵-۳-۲- جناح و تمدن‌های مختلف

وجود چندین جناح و تمدن مخلف در یک بازی استراتژی می‌تواند باعث جذب شدن بازیکنان به بازی شود. تمدن‌های مختلف که در بازی در جنگ چندین ساله هستند باعث شکل گیری جناح‌های مختلف بازیکنان

می‌شود و آن‌ها را ترقیب می‌کند تا به بازی برگشته و به کمک تمدنی که علاقه بیشتری به آن دارند سعی در نابودی تمدن‌های رقیب بکنند که مثال آن در بازی World Of Warcraft به چشم می‌خورد که با وجود انتشار بازی در ۱۷ سال پیش بازیکنان هنوز به بازی کردن در جناح مورد علاقه خود(هورد یا آلاینس) کرده و در جنگ‌های مقابل هم این دو جناح شرکت کنند و در نتیجه بازی را به فروش بالای خود برسانند.

۵-۳-۳- رعیت و مردم عادی در پادشاهی

وجود مردم عام در پادشاهی بازیکن باعث می‌شود تا بازیکن علاوه بر سعی در جمع کردن منابع و حمله به رقیبان چالش راضی نگه داشتن مردم عام را نیز داشته باشد و بازی کردن را همراه با سخت‌تر کردن آن جذاب‌تر نیز بکند.

۵-۳-٤- گروه بندی و ساخت کلن

وجود کلن ها در پروژه باعث می‌شود بازیکنان به مقابله و جنگ‌های بین کلنی برای بالا بردن رتبه کلن خود بشوند و آن‌ها را ترقیب به بازی بیشتر می‌کند.

۵-۳-۵- پرداخت درون برنامه‌ای

سیستم پرداخت درون برنامه سودآوری پروژه را تضمین می‌کند، به کمک آن بازیکنان برای پیشی گرفتن از هم به خرید پول داخل برنامه پرداخته و باعث می‌شود پروژه نیازی به سیستم خرید برنامه نداشته باشد و کاربران بیشتری را به آن جذب کند.

۵-٤- اهمیت‌ها و چالش‌ها

در این قسمت به اهمیت‌ها و چالش‌های سر راه پروژه می‌پردازیم و آن‌ها را بررسی می‌کنیم.

۵-٤-۱- اهمیت‌ها

ساخت بازی‌های رایانه‌ای ایرانی به صورت کلی کسب بازار از شرکت‌های خارجی و خارج نشدن ارز بر اثر آن است. به تازگی رییس بنیاد ملی بازی‌های رایانه‌ای اعلام کردند که ایران دارای ۳۲ میلیون کاربر بازی‌های

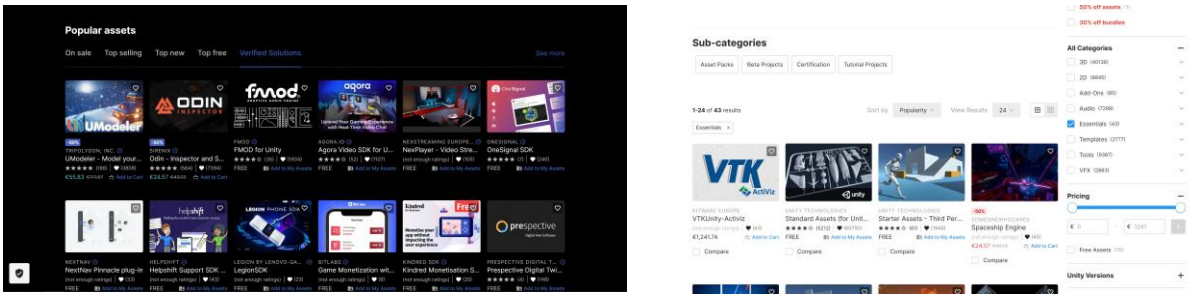
رایانه‌ای می‌باشد، که در ۱۰ سال گذشته دو برابر شده است. این تعداد بازیکن ایرانی اگر خریدهای بازی‌های خود را از فروشگاه‌ها و شرکت‌های خارجی انجام ندهند باعث ماندن ارز در کشور می‌شود. در سخنان رییس بنیاد ملی بازی‌های رایانه‌ای به پرمخاطب‌ترین ژانرها نیز اشاره شد که پس از ژانر ورزشی ژانر استراتژی می‌باشد. در سال ۱۳۹۸ در ایران ٤۳۰۰ میلیارد تومان صرف بازی شد که این مبلغ شامل هزینه‌های بازیکنان روی سه پلتفرم موبایل، کنسول و رایانه می‌شود. از این ارقام به‌سادگی می‌توان دریافت که ما با صنعت بسیار بسیار مهمی مواجه‌ایم که ابعاد اشتغال‌زایی و تولیدی وسیعی دارد.

در پروژه ما اگر حمایت سرمایه‌گذاران در آن باشد می‌توان از برنامه نویسان و طراحان در آن بهره برد و می‌تواند سرمایه مهمی از کشور را از این پروژه که دومین ژانر پرمخاطب در ایران است را حفظ کرد و اشتغال‌زایی نمود و درآمد ارزی را نیز درصورت باکیفیت بودن آن به دست آورد.

۵-٤-۲- چالش‌ها

بازی‌ها و به طور کلی برنامه‌ها پس از انتشار اولیه آپدیت‌هایی نیز منتشر می‌کنند همچنین فروش و انتشار بازی‌ها برای هر پلتفرم فروشگاه‌ها و سایت‌های مخصوص خود را دارند که شرکت‌های سازنده پس از ساخت محصول خود را در آن منتشر کنند و عملیات فروش و تبلیغات را سرعت بخشند برای مثال برای پلتفرم اندروید فروشگاه بازار و مایکت وجود دارد ولی برای پلتفرم رایانه شخصی در ایران این پلتفرم وجود ندارد که برای بازی‌ها در دیگر کشورها فروشگاه‌هایی مانند Steam وEpic Games وجود دارد ولی از آنجا که ایران در تحریم جهانی است این فروشگاه‌ها در دسترس توسعه دهندگان ایرانی نیست. در نتیجه این فقدان فروشگاهی برای فروش ، تبلیغ و انتشار نسخه‌های جدیدتر از جمله چالش‌های این پروژه است.

یکی دیگر از چالش‌ها که باعث آن تحریم‌ها می‌باشند بسته بودن سایت یونیتی و Asset Store است که باعث می‌شود گرفتن نسخه‌های جدید یونیتی مشکل بوده و اضافه کردن کتابخانه‌ها و Assetها بسیار مشکل باشد و خرید آن به صورت مستقیم از Asset Store ممکن نباشد. در زیر تصاویری از این سایت Asset Store که از متعلقات خود یونیتی می‌باشد را مشاهده می‌کنیم.



شکل ۳۸) فروشگاه خرید متعلقات یونیتی (Asset Store)

یکی دیگر از چالش‌ها حمایت نکردن بازار و دولت از سازندگان بازی داخلی می‌باشد که به دو صورت است که در زیر شرح می‌دهیم.

۱- فروش بازی‌ها و برنامه‌های ایرانی به صورت رایگان و سرقت اینترنتی در سایت‌های مختلف ایرانی که باعث می‌شود حمایت مورد نیاز از سازنده انجام نشود که دولت و قانون عملا برای جلوگیری از آن اعمال زیادی را انجام نمی‌دهند.

۲- نبود حمایت دولت به صورت مالی که باعث می‌شود سرمایه اولیه برای ساخت بازی‌ها وجود نداشته باشد و استارت‌آپ‌ها سرمایه مورد نیاز را نداشته باشند.

در لایحه بودجه ۱۴۰۱ که از سوی دولت تقدیم مجلس شده، موضوع دریافت عوارض ده درصدی از درآمد حاصل از نشر و فروش بازی‌های ویدیویی خارجی برای سال دیگر هم تمدید شده است؛ مبلغی که قرار است به دست بنیاد ملی بازی‌های رایانه‌ای برسد و صرف تولیدات داخلی بشود. سال ۹۷ بود که مرکز ملی فضای مجازی در راستای تعریف قانون بودجه مصوبه‌ای را تصویب کرد که در آن عبارت «دستورالعمل اجرایی دریافت ۱۰ درصد از درآمدهای حاصل از نشر و عرضه تجاری و رسمی بازی های رایانه ای خارجی» ذکر شده بود. در همان زمان حواشی زیادی بر سر این عوارض به وجود آمد و بازی‌سازان زیادی با آن مخالفت کردند اما این قانون اجرایی شد و تا امروز ادامه دارد. برنامه بودجه ۱۴۰۱ نیز حکایت از این دارد که این مساله در سال بعد نیز تمدید شده است. کارشناسان معتقد هستند که این مساله در نهایت به نفع بازار داخلی نیست و ریزش کاربر در این حوزه به وجود خواهد آمد؛ این مخالفین معتقد هستند محل هزینه این پول‌ها نیز به طور دقیق مشخص نیست. از آنسو عده‌ای معتقدند بازار بکر بازی‌های رایانه‌ای در داخل کشور نباید رها باشد و با اخذ عوارض در آن می‌توان برای توسعه این صنعت هزینه کرد. موافقین این مساله معتقد هستند بازار ایران متاثر از اقتصاد دولتی

است و بازار بازی‌های ویدیویی نیز از این قاعده خارج نیست. در هر حال چیزی که مشخص است این مساله باعث افزایش قیمت بازی‌های خارجی می‌شود و این موضوع احتمالا به ضرر مصرف کنندگان بازار است. شاید مواردی نظیر معافیت‌های مالیاتی و... راه‌حل‌های بهتری برای حمایت از بازی‌های داخلی که توانستند در سالهای اخیر رشد کنند باشد.

۵-٤- نتیجه‌گیری و پیشنهادات

در این پروژه به ساخت یک بازی استراتژی زمان واقعی به کمک موتور یونیتی و زبان سی‌شارپ پرداختیم. در این پروژه موفق به ساخت این پروژه شدیم و به کمک پلاگین Mirror و Steam بازیکنان را به یکدیگر در رقابت آنلاین متصل کردیم. همچنین سیستم مسیریابی نیروها که در موتور یونیتی در Navigation به کمک الگوریتم *A پیاده سازی شده است را استفاده کردیم.

به طور کلی سیستم‌های پیاده سازی شده در این پروژه را نام برد.

۱- حرکت نیروها ۲- ساخت نیروها و ساختمان‌ها ۳- انتخاب نیروها ٤- دستور به نیروها ۵- ذخیره نیروها ٦- هدف گیری نیروها ۷- حمله کردن نیروها ۸- سلامتی نیروها و ساختمان‌ها ۹- برد و باخت ۱۰- منابع ۱۱- صف ساخت نیروها ۱۲- حرکت دوربین ۱۳- نقشه مسابقه ١٤- لابی ۱۵- رقابت آنلاین توسط Steam

در پایان پیشنهاداتی برای بهتر شدن این پروژه که در بخش نکات قابل بهبود پروژه بررسی کردیم را در زیر نام برده و کمی توضیح داده‌ایم.

اضافه کردن بخش داستانی و کمپین: در اضافه کردن این بخش می‌توانیم از داستان‌های تاریخ ایران و جنگ‌های بزرگ کشورمان استفاده کنیم و به این صورت تاریخ کشورمان را به کمک این پروژه به کاربران آموزش دهیم.

اضافه کردن جناح و تمدن‌های مختلف: در اضافه کردن آن می‌توانیم در لابی به بازیکنان فرصت انتخاب تمدن‌ها و جناح‌های مختلف را بدهیم برای مثال بازیکنان بتوانند بین جناح حکومت هخامنشیان و یونانیان دو جناح را انتخاب کرده و با هم به مسابقه بپردازند. این پیشنهاد نیاز به وجود مدل سازان و طراحان ماهر دارد تا ساختمان‌ها و نیروهای جناح‌ها و تمدن‌ها را بسازند.

وجود رعیت و مردم عادی: برای اضافه کردن آن میتوان ابتدا چند نمونه ساختمان که خانه‌ها را نمایندگی می‌کند قرار داد که در آن رعیت‌ها ساخته شده و سپس بازیکنان، سربازان و کارگران را با تعلیم رعیت‌ها به دست آورند.

گروه‌بندی و ساخت کلن: برای ساخت این سیستم می‌توانیم در منو اصلی بخشی برای یافتن بازیکنان قرار دهیم و بازیکنان با انتخاب آنان و ساخت کلن گروه خود را ساخته و با جنگ با دیگر گروه‌ها بپردازند.

پرداخت درون برنامه‌ای: در اضافه کردن این سیستم می‌توانیم ابتدا به بازیکنان یک تمدن داده و بقیه تمدن‌ها را برای به دست آوردن در قسمت فروشگاه برنامه به فروش بگذاریم تا بازیکنان برای دست‌یابی به آنان پرداخت‌هایی را انجام دهند و پس از پرداخت بتوانند از تمدن خریداری شده در مسابقات خود استفاده کنند.

واژه‌نامه

استراتژی زمان‌واقعی..... Real time strategy

استراتژی مبتنی بر نوبت..... Turn based strategy

اقدام..... Action

بازی نقش‌آفرینی..... Role-playing game

بازی مدینه فاضله..... Utopia game

بازی شهرسازی..... City building game

بازی خدا..... God mode game

بازی تله ماسه..... Dune Game

بازی دالاس..... Dallas Game

بیشترین متصلین..... Max connections

بسته الحاقی..... Dlc

پلتفرم..... Platform

پیش‌ساخته..... Prefab

پلتفرم استیم..... Steam

پایگاه..... Base

تجربه کاربری..... User Experience

چندنفره آنلاین..... Multiplayer online

حامل..... Transport

خط لوله همگانی..... Universal render pipeline

دیدگاه ایزومتریک..... Isometric view

راه‌یابی..... Navigation

رویداد اجرا شده توسط یونیتی..... UnityEvent

رابط کاربری..... User interface

زبان مدل‌سازی یکپارچه..... Unified Modeling Language(UML)

زبان برنامه سازی سی‌شارپ..... C# programing language

سالن انتظار..... Lobby

ساختمان..... Building

سرور–موکل..... Client-Server

شبکه..... Network

شبکه محلی..... Local Area Network

شی..... Object

صفحه اصلی..... Main menu

فروشگاه متعلقات یونیتی..... Unity Asset Store

کتابخانه میرور..... Mirror library

کمپین..... Campaign

گانت چارت..... Gantt Chart

گیم‌پلی..... Gameplay

موتور بازی سازی یونیتی..... Unity game engine

منابع..... Resources

محیط رومیزی..... Desktop

مولفه..... Component

مکان..... Position

مقصد..... Destination

مدیر شبکه..... Network manager

نقشه کوچک..... Minimap

نمودار رابطه–موجودیت..... Entity-Relationship Diagram (ERD)

نمودار جریان داده ها..... Data-Flow Diagram(DFD)

نمودار حالت..... State diagram

نمودار کلاس..... Class Diagram

نمودار مورد کاربرد..... Use case diagram

نمودار توالی..... Sequence diagram

نمودار ارتباطی..... Communication diagram

واحد..... Unit

فهرست منابع

1. Bruce Geryk, Archived from the original on March 31, 2008" ,A History of Real-Time Strategy Games" GameSpot

2. Adams, Dan (April 7, 2006). "The State of the RTS". IGN. Archived from the original on April 9, 2006.

3. Barry, Tim (May 11, 1981). "In Search of the Ultimate Computer Game". InfoWorld. pp. 11, 48. Retrieved April 17, 2019.

4. Moss, Richard (September 15, 2017). "Build, gather, brawl, repeat: The history of real-time strategy games". Ars Technica. Retrieved October 20, 2017.

5. "The evolution of gaming: computers, consoles, and arcade". Ars Technica. October 11, 2005.

6. Weiss, Brett (2011). Classic Home Video Games, 1972–1984: A Complete Reference Guide. McFarland & Co. p. 291. ISBN 9780786487554.

7. Loguidice, Bill; Barton, Matt (2012). Vintage Games: An Insider Look at the History of Grand Theft Auto, Super Mario, and the Most Influential Games of All Time. CRC Press. p. 73. ISBN 9781136137587.

8. "Archived copy". www.allgame.com. Archived from the original on November 14, 2014. Retrieved January 15, 2022.

9. Barton, Matt. "The History of Computer Role-Playing Games Part 2: The Golden Age (1985-1993)". Gamasutra. Retrieved October 16, 2017. SSI's most famous non-CRPG game is probably Cytron Masters (1982), one of the first (if not the first) real-time strategy games.

10. Scott Sharkey. "Hail to the Duke". 1UP.com. Archived from the original on September 13, 2004. Retrieved March 1, 2011.

11. Sartori-Angus, Alan (December 1982). "Cosmic Conquest". BYTE. pp. 3, 124. Retrieved October 19, 2013.

12. "RTSC Historical RTS List". Archived from the original on August 23, 2006. Retrieved August 5, 2006.

13. Bokosuka Wars (translation), Nintendo

14. Dru Hill: The Chronicle of Druaga Archived 2005-01-19 at archive.today, 1UP

15. Sega Ages: Gain Ground, IGN, July 20, 2004

16. Top 10 Renovation Games, IGN, June 17, 2008

17. Herzog Zwei, GameSpy

18. "The Making of... Dune II". Edge. Next-Gen.biz. December 9, 2008. Retrieved July 27, 2011.

19. Clarke-Willson, Stephen (August 18, 1998). "The Origin of Realtime Strategy Games on the PC". The Rise and Fall of Virgin Interactive. Above the Garage Productions. Archived from the original on May 4, 2003. Retrieved January 30, 2012.

20. "The Essential 50 Part 31: Herzog Zwei". Archived from the original on September 13, 2004. Retrieved December 17, 2006.

21. Walker, Mark. "Strategy Gaming: Part I – A Primer". GameSpy. Archived from the original on August 10, 2010. Retrieved October 28, 2007.

22. "The History of Command & Conquer". NowGamer. Retrieved May 14, 2014.

23. "A Brief History Of Strategy Games | Starborne". starborne.com.

24. Rollings, Andrew; Ernest Adams (2006). Fundamentals of Game Design. Prentice Hall.

Abstract

Worlds At War: An online real-time strategy game

Real-time strategy is a subset of strategy video games in which players do not play in turns and actually allow all players to play in "real-time" simultaneously. In our country, Iran, due to the large and wide market that computer games have and strategy games, which are one of the most popular genres, so it is necessary to use this market. We will first explain the real-time strategy and its history, and then explain several examples made by large companies, and then compare them. In this project, we analyze a real-time strategy game, then present its diagrams, and then, with the help of the Unity game engine, which is one of the most famous game development programs, the C# programming language and Mirror, which is used to connect games to the network in Unity, we implement this project. At the end, we compare the obtained results with the mentioned examples, then we mention the improvements of the project, and then we draw the final conclusion and suggestions.

Keywords: Mirror, Real Time Strategy, C # Programming, Object Oriented Programming, Algorithm A*

MAHALAT INSTITUE OF HIGHER EDUCATION

Faculty of Engineering - Department of Computer

((B. Sc.)) Thesis on Software Engineering

Subject:

Worlds At War: An online real-time strategy game

Thesis Advisor:

Dr. Saeed Doostali

By:

Nima Sattari

Winter 2021