Семинарски рад Ц развој компјутерских игара

Никола Радојчић 62/19 РН

22. мај 2022.

Садржај

1 Увод

У овом семинарском раду биће описана идеја и имплементација стратешке игре направљене у *Unity* алату. Игрица се састоји искључиво од своје кампање која има структуру која је комбинација између кампања из игрица *Faster than Light* и *Homeworld* где су саме мисије сличне мисијама у игрици *Stroghold*. Између мисија у кампањи се очувавају сви војници и ресурси који су у тој мисији сакупљени као у *Homeworld*, док се мисије бирају на мапи на којој играч има избор путање коју жели да узме да би стигао до циља. Саме мисије се састоје од постављања кампа експлоатације ресурса изградње помоћних зграда као и тренирања војника зарад победе над непријатељским снагама које се већ налазе на мапи пре доласка или се појављују после одређеног времена. Непријатељ се не развија током мисије. Игрица би требала да се дешава у средњем веку као у *Stroghold*.

2 Unity

Unity је алат за развој видео игара. Он служи за бржи и лакши развој видео игара. Пошто су основе потребне за игре већ написане, нема потребе да се оне поново праве при започињању развоја сваке нове игре, самим тим и развој игара постаје много бржи и јефтинији.

2.1 Сцене

Сцена садржи све остале елементе. Сценама се представљају различити нивои и менији у игрици. Нова сцена почиње само са објектом за камеру па се у њу убацују остали елементи који су потребни у тој сцени.

2.2 Камера

Камера одређује поглед који ће корисник имати на простор изграђен у тој сцени. Померањем камере се мења оно шта корисник може да види. Камера може бити ортографска и перспективна. Перспективна камера имитира начин на који људи виде свет. Код перспективне камере идеја је да се лакше може одредити удаљеност нечега, то јест што су ствари даље од камере оне изгледају мање и уже од оних које су ближе. Овај тип камере се најчешће користи за игрице у три димензије. Ортографска камера све ствари приказује у својој величини не умањујући их у зависности од раздаљине, зато се она најчешће користи у игрицама у две димензије

2.3 Платно

Платно(енг. *Canvas*) служи за израду корисничког интерфејса. Сви елементи платна су увек видљиви на погледу који камера приказује и увек га прате. На њему се налазе све иконице и дугмићи који су део сцене али не и одређене локације у нивоу који та сцена представља. Платно се аутоматски поставља и развлачи преко простора екрана који заузима приказ са камере.

2.4 Извор Звука

Извор звука(енг. *Audio Source*) пушта клип који му се додели. Звук који долази из извора звука може да губи гласноћу у зависности од раздаљине, то јест да постаје тиши и тиши све док не престане да се чује када се слушалац звука(енг. *Audio Listener*) удаљи предалеко. Извор звука се може доделити као компонента било ком објекту у сцени.

2.5 GameObject

У једној сцени сваки елемент је заправо *GameObject* који садржи одређене компоненте. Компоненте које он садржи одређују шта он ради. Сваки *GameObject* почиње са *transform* компонентом која одређује његову позицију, ротацију и омогућава његово скалирање.

2.6 Prefab

Prefab омогућава да се неки објекат једном изради и сачува, па да се убудуће само додаје из тог сачуваног. Он омогућава и да се направе истовремене измене на свим објектима који потичу из њега, тако што се измене начине над њим. Онда се сачувају па се аутоматски прошире на све остале.

3 Кампања

Кампања ће се одржавати на мапи на којој постоји пут. Сваки пут води до неког чвора на ком ће се одиграти мисија. Чвор може представљати једну од два типа мисија (битка на пољу или интерактивна мисија).

3.1 Битка на пољу

Овај тип мисије се одиграва тако што је део непријатељске војске већ на мапи обично чувајући неке ресурсе. Играч треба да победи непријатеље који се налазе на тој мапи.

3.2 Интерактивна мисија

Овај тип мисије се одиграва у целости на кампањској мапи где играч добија избор одлука које ће утицати на кампању на различите начине, као на пример може да жртвује брзину кретања до следећег чвора али да добије неке ресурсе за узврат.

4 Мисија

Све мисије, осим интерактивних, се одигравају тако што се сви играчеви војници појављују на једном делу мапе заједно са кампом. Ако зграда која представља камп буде уништена играч је изгубио игру. Сви војници и радници се праве у кампу и баракама а захтевају популацију и ресурсе. Ресурсе скупљају радници. Популација зависи од до сада сакупљене популације у кампањи и неможе се повећати током мисије осим у неким специјалним случајевима или распуштањем неког војника или радника. Када се направи војник или радник он троши популацију која се неће вратити ако он умре.

5 Приказ игре

Током гледања кампањске мапе она треба да изгледа као старе мапе тог периода, то јест претежно се концентришући на путеве, насеља и битне локације, који су представљени чворовима, док остатак мапе приказује свет али нацртан отприлике у односу на чворове.

Током мисија играч види уближену мапу која представља терен на том чвору. Ствари на мапи су приказане одговарајућим сличицама.

6 Јединице

Све јединице у игрици имају исте података који их карактеришу а разликују се само по вредностима. Јединице можемо поделити у две категорије на основу типа популације који је потребан за њихово регрутовање. Постоје јединице прве и друге класе, јединице прве класе долазе од популације ратника а јединице друге класе долазе из дела популације способне за борбу. Свака јединица може припадати или играчу или непријатељу, и она има своју цену која представља потребну опрему за опремање те јединице. Свака јединица може да се туче али њена ефективност у тучи зависи од њене јачине и способности пробоја оклопа у тучи. Неке од јединица имају способност борбе на даљину. У том случају ако нису у контакту са непријатељем и ако се непријатељ налази у њиховом домету они за напад не користе своје вредности за борбу у тучи, већ се ослањају на своју јачину и пробој оклопа на даљину, а прелазе на вредности за тучу при контакту са непријатељем. Јединице које гађају имају и своју брзину гађања као и прецизност која се користи при прорачунима да би се одредило да ли јединица чини икакву штету. Свака јединица има вредности за оклоп одбрану и своје здравље. Оклоп директно смањује штету коју наноси непријатељска јединица, то јест он се одузима од непријатељског напада и ако та вредност падне на нулу јединица неће претрпети никакву штету. Пре него што се оклоп одузме од непријатељског напада, непријатељски пробој оклопа умањује оклоп по следећој формули:
о*(1-по/100)
о-оклоп
по-пробој оклопа
Одбрана умањује непријатељски напад на исти начин ко што пробој оклопа умањује оклоп али она умањује непријатељски напад тек када се од њега одузме оклоп. Здравље јединице одређује колико штете јединица може да претрпи. Када здравље дође до нуле јединица умире. Штета се рачуна тако што се од здравља одузме непријатељски напад после свих пређашњих прорачуна. Постоји седам врста јединица, а то су: радници, копљани, мачеваоци, стрелци, праћкаши, аркуебусијери и коњаници.

6.1 Јединице друге класе

Јединице друге класе чине радници, копљаници, праћкаши и аркуебусијери. Само радници могу да сакупљају ресурсе и да граде. Радници су најпростији тип јединица они не могу да гађају а уједно су и најгори у тучи, али зато су најјефтинији и намењени су за скупљање ресурса. Копљаници су намењени за тучу али су најслабији у том погледу ипак они су и даље јефтини и представљају једину јединицу намењену за тучу у јединицама друге класе. Праћкаш је најјефтинија јединица која може

да гађа. Њихов напад је слаб и немају никакав пробој оклопа али имају највећи домет. Аркубусијери користе примитивне пушке које јако споро пуцају и имају очајну прецизност па самим тим и мали домет, тако да ова јединица има најгори домет, прецизност и брзину гађања. Такође су веома скупи због цене пушака, али због лакоће тренирања они спадају у другу класу и пошто користе пушке имају највећи пробој оклопа од свих јединица. Њихов пробој оклопа је толико велик да они игноришу непријатељски оклоп. Све јединице са способношћу гађања могу да гађају преко других јединица и брину само о свом домету.

6.2 Јединице прве класе

Јединице прве класе чине мачеваоци, стрелци и коњаници. Јединице које припадају другој класи долазе од ратника и не може им се наредити да сакупљају ресурсе, самим тим они немају никакву сврху ван борбе. Мачеваоци су веома добри у тучи имају јак оклоп и добру одбрану, они поред коњаника поседују највећи напад и пробој оклопа захваљујући резервном оружју али имају и најмању брзину у односу на све јединице и уједно су друга најскупља јединица. Стрелци имају способност гађања и имају највећу брзину и прецизност гађања док су њихови напад и пробој оклопа на даљину мањи од оног који аркубусијери поседују. Они ће засигурно у одређеном временском периоду нанети више штете спрам аркубусијера. Стрелци су најбољи у тучи од свих јединица са способношћу гађања захваљујући њиховој вештини. Коњаници су најбоља јединица у тучи имајући све предности мачеваоца а још и највећу брзину као и највеће здравље у односу на све јединице, али су зато и најскупља јединица у игрици.

6.3 Подаци о јединицама

Тип	Рад	Коп	Мач	Стр	Пра	Арк	Коњ
Ц	50	100	200	100	75	200	400
НТ	2,5	10	20	4	2,5	2,5	25
ПОТ	0	0	50	0	0	0	50
НД	0	0	0	8	2,5	10	0
ПОД	0	0	0	70	0	100	0
ПРЕ	0	0	0	90	70	40	0
ДОМ	0	0.5	0	8	10	5	0
БГ	0	0	0	0.25	0.5	1	0
ОК	0	0	7	0	0	0	7
ОД	0	10	50	20	0	0	50
ЗДР	25	25	50	50	25	25	75
БРЗ	1	1	0,3	1	1	1	3

Табела 1: Ц - Цена, НТ- Напад за тучу, ПОТ - Пробој оклопа у тучи, НД - Напад на даљину, ПОД - Пробој оклопа на даљину, ПРЕ - Прецизност, ДОМ - Домет, БГ - Брзина гађања, ОК - оклоп, ОД - Одбрана, ЗДР - Здравље, БРЗ - Брзина

7 Упутство за играње

7.1 Кампањска мапа

На кампањској мапи играч почиње на првом чвору при дну екрана и од њега може да иде само за један чвор напред. При клику на наредни чвор мисија која одговара том чвору ће се започети.

7.2 Интерактивна мисија

Интерактивне мисије почињу када се кликне на њихов чвор. На почетку ове мисије промениће се позадина и исписати текст који описује догађај који се одиграо и добија се избор између три опције. Нека од опција мора бити одабрана да би се наставило даље. Када се одабере нека од три опције исписује се текст који описује последице тог избора и испод њега саме последице по играча. Када играч прочита текст може да стисне дугме за назад да би се вратио на кампањску мапу.

7.3 Мисија на пољу

У овим мисијама играч треба да скупља ресурсе са радницима тако што одабере радника и кликне на ресурс. Када скупи довољно ресурса играч може да нареди раднику да направи зграду.

7.3.1 Прављење зграда

Зграде се праве тако што се одабере радник и онда у доњем левом углу се одабере зграда која ће се градити. Одабиром зграде за изградњу прелази се у режим грађења. У режиму грађења више се неможе ништа друго одабрати, а са десним кликом се поставља зграда. Из режима грађења се излази на леви клик. Када је зграда постављена треба послати радника да је изгради, тако што се одабере радник и стисне се десни клик на зграду.

7.3.2 Тренирање јединица

Када се одабере зграда која може да тренира јединице, у доњем левом углу појави се одабир јединица за тренинг. Јединице се у зградама тренирају кликом на њихову иконицу, после чега се аутоматски додају у ред тренирања те зграде. Јединице се могу избацити из реда прављења зграде кликом на иконицу која се појавила доле у средини када су додате у ред. Док је одабрана зграда десним кликом се може одабрати место на ком ће се јединице скупљати кад се направе.

7.3.3 Командовање јединицама

Јединице се могу одабрати превлачењем, где ће предност имати јединице, и кликтањем на јединицу или зграду. Када су неке јединице већ одабране држањем дугмета *CTRL* се могу додати и друге јединице у ту селекцију. Када постоје одабране јединице десним кликом се њима поставља локација на коју да иду, ако се кликне на непријатељску јединицу јединице ће ићи да је нападну, и стаће на ивици свог домета. Јединице које гађају стају само ако им је наређено да нападају у супротном они ће ићи исто директно на локацију ко и остале јединице. Зграде такође нападају.

8 Одабир објеката

Одабир јединица се обавља преко три класе *PlayerManager*, *MultiSelect* и *InputHandler*. Класа *PlayerManager* садржи промењиву која показује на над објекат свих играчевих јединица и позива током сваког фрејма функцију `HandleUnitMovment()` из класе *InputHandler*. *MultiSelect* је помоћна класа која у себи садржи функције које се касније користе при одабиру више јединица превлачењем. Улаз од корисника се обрађује у класи *InputHandler*. Функције `SelectUnit(Transform,bool)` и `DeselectUnits()` у класи *InputHandler* одрађују одабир већ одређеног објекта и брисање пређашњих одабира респективно. Одабир већ одређеног објекта у функцији `SelectUnitTransform,bool` се одвија тако што се прво провери да ли је други параметар тачно или нетачно и у случају да је нетачно позива се функција за брисање пређашњих одабира јер други параметар говори да ли смеју да се одаберу више објеката. После провере другог параметра прослеђена трансформација која представља локацију јединице се додаје у глобалну промењиву која чува податке о одабраним објектима. На крају се у одабраном објекту проналази његова *Interactable* компонента било то *IUnit* или *IBuilding* и позива се њена функција `OnInteractEnter()` . Када се пређашњи одабири бришу по позиву функције `DeselectUnits()` пролази се кроз листу пређашњих одабира и сваком се позива функција `OnInteractExit()` из њихових компоненти, на крају се избрише све из те листе.

8.1 Одређивање објеката за одабир кликтањем

При притиску левог дугмета на мишу у функцији `HandleUnitMovment()` испуњава се услов прве петље и у њој се онда преузима локација миша и потом претвара у одређену тачку у свету игрице. Потом се проверава да ли се неки објекат налази на тој тачци. Ако се на тачци налази објекат позива се функција за одабир јединица и прослеђује му се као први параметар трансформација погођеног објекта, а као други параметар резултат провере да ли је једно од *Control(Ctrl)* дугмади притиснуто. Ако функција за одабир јединица неможе да нађе *IUnit* компоненту позива се функција за додавање зграда и прослеђује јој се трансформација погођеног објекта. Ако ни један од жељених објеката није погођен бришу се сви стари одабири као и у случају да ништа није погођено.

8.2 Одређивање јединица за одабир превлачењем

Када год се притисне лево дугме миша увек се једна глобална промењива поставља на тачно а када се то дугме пусти промењива се поставља на нетачно. До год је та промењива постављена на тачно исцртава се на екрану правоугаоник од почетне до тренутне тачке помоћу класе

MultiSelect. Када се дугме пусти пролази се кроз сву децу над објекта који се налази у *PlayerManager* и онда кроз сву децу те деце што представља јединице. Затим се проверава да ли се та јединица налази у распону правоугаоника, ако се налази позива се функција за одабир објекта и прослеђује се та јединица и тачно. На крају се промењива враћа на нетачно.

8.2.1 Цртање правоугаоника и одређивање његове границе

Цртање правоукаоника који означава селекцију и одређивање његових граница при селекцији се врши у статичкој класи *MultiSelect*.

Функција `DrawScreenRect(Rect, Color)` црта задати правоугаоник у задатој боји.

`DrawScreenRectBorder(Rect, float, Color)` црта ивице правоугаоника тако што на свакој од четири стране нацрта танке правоугаонике који покривају целу страну.

`GetScreenRect(Vector2, Vector2)` прима два вектора и враћа правоугаоник који је одређен њима.

`GetViewPortBounds(Camera, Vector2, Vector2)` враћа границе које одређују два прослеђена вектора.

9 Кретање

За проналажење пута користи се *A* Pathfinding Project*. Овај додатак обрађује мапу и проналази пут од објекта до његовог циља избегавајући препреке. Свака јединица има компоненте *AIPath* и *AIDestinationSetter* које одређују дестинацију и остале параметре везане за кретање, њих обезбеђује додатак. Класа *VectorDestinationSetter* користи постојеће функције из наведених компоненти да би задала дестинацију користећи прослеђени вектор дестинације или трансформацију и зауставну раздаљину од мете. Ова класа садржи и функцију за брисање дестинације као и функцију за постављање брзине. Све функције ове класе су тривијалне и обично само позивају одговарајуће функције из компоненти пазећи да компонента *AIDestinationSetter* нема задату мету осим ако се преко ње не задаје дестинација јер би она у супронтом преправљала дестинацију на ону која њој одговара.

У класи *InputHandler* функција `HandleUnitMovment()` обрађује кретање јединица тако што у својој трећој ако петљи проверава да ли је притиснут десни клик и да ли постоје одабране јединице. Ако нема одабраних јединица а има одабрана зграда онда само позива функцију за постављање тачке окупљања. Ако постоје одабране јединице проверава се који је слој погођен и прослеђује се трансформација погођеног објекта ако је он валидна мета за јединицу у супротном вектор који представља

гађану локацију функцији `MoveUnit(Transform, bool = true, bool = false)` ili `MoveUnit(Vector2)` класе *PlayerUnit*.

`MoveUnit(Vector2)` прима вектор који представља дестинацију на коју јединица треба да оде и онда ту дестинацију прослеђује класи *VectorDestinationSetter* и поставља вредност промењиве `hasTarget` на нетачно јер јединица више нема мету у виду другог објекта већ иде до одређене позиције. Потом пушта адеквата звук који представљ одговор на наређење за покрет.

`MoveUnit(Transform, bool = true, bool = false)` prima transformaciju објекта који је мета и две опционе промењиве које говоре да ли је мета пријатељски објекат и да ли је мета ресурс респективно. Прво се проверава да ли је мета пријатељска, ако јесте поставља се дестинација на њу и проверава се да ли тренутна јединица има способност поправљања, ако има онда се дата трансформација поставља као мета и промењива `isRepairing` се поставља на тачно, потом се излази из функције. Ако мета није пријатељски објекат проверава се да ли је она ресурс и да ли тренутна јединица може да ради то јест да скупља тај ресурс помоћу промењиве `canWork`. Ако су оба упита друге провере тачна промењива `isMining` која обавештава да ли јединица копа се поставља на тачно, трансформација се поставља за мету и дестинацију па се потом излази из функције. Ако ни једна од прве две провере нису тачне то значи да је мета непријатељска јединица или зграда. Сада функција поставља трансформацију за мету и прослеђује ју као дестинацију заједно са својим дометом који представља раздаљину на којој јединица треба да стане. Промењива `hasTarget` се поставља на тачно и преузима се компонента `EnemyUnit` од те јединице за коришћење током борбе. На крају се пушта одговарајући звук за примање наређења.

10 Борба и Рад

Функције за прорачун борбе се налазе у статичкој класи *Combat*, док рад и поправљање зграда се прерачунавају у класама јединица, ресурса и зграда. Сва борба почиње од функција које се налазе у абстрактној класи *SharedUnit*.

`CheckForEnemyTargets(bool)` прима једну промењиву која говори да ли је јединица која ју зове играчева. Ова функција прво проверава да ли постоји неки *collider* у кругу са центром у јединици и полупречником који одговара раздаљини на којој јединице почињу да нападају и налази се у глобалној промењивој `aggroDistance`. Потом се проверава који је индекс *collider*-а који је најближи од пронађених и од тада се ради са објектом којем је он придружен. Када се пронађе најближи објекат проверава се да ли јединица већ има мету, јер ако има онда се не поставља нова, у супротном се проверава да ли је јединица играчева или не. Ако јединица није играчева онда узима компоненте које одговарају играчу од

најближег објекта а у супротном оне које одговарају непријатељу. Ако јединица нема мету онда она поставља најближи објекат као своју мету и постави промењиву `targetUnit` на вредност компоненте за јединице од најближег објекта, било то *PlayerUnit* или *EnemyUnit*. Ако `targetUnit` није добио вредност компоненте за јединице од најближег објекта то значи да она не постоји јер је тај објекат зграда па ју поставља на вредност компоненте за зграде најближег објекта.

`TakeDamage(float, int)` прима количину нанете штете и пробој оклопа непријатеља и додељије вредност здравља свом тренутном здрављу које враћа функција истог имена из класе *Combat* када јој се проследе нанета штета и пробој оклопа непријатеља као и оклоп, одбрана и тренутни хелти јединице. Потом се пушта клип који представља рањавање јединице.

`Repair()` функција ради прорачун поправки она ради слично ко и *Attack* функција из класе *Combat* само што директно позива функцију истог имена из класе *PlayerBuilding* и пушта звук за поправку.

`Work()` функцију има само *PlayerUnit* класа и она ради слично као и прошла функција само што ради са ресурсима.

`Attack()` функција прво проверава да ли мета и даље постоји јер је могла бити уништена од стране других јединица. Ако не постоји поставља се `hasTarget` на нетачно и дестинација се уклања па се потом излази из функције. У супротном прерачунава се раздаљина од мете. У привремену промењиву се записује повратна вредност коју враћа промењива истог имена из класе *Combat* када јој се проследи тренутно време чекања до следећег напада, раздаљина од мете, време чекања између напада, компонента од задужена за примање штете од мете, и основни подаци о јединици. Ако је та функција вратила нешто веће или једнако нули пушта се звук за пуцање или борбу у зависности да ли је раздаљина већа од 2. Ако је функција вратила -2 то значи да непријатељ више не постоји и онда се ради исто што и на почетку ако нема непријатеља.

10.1 Класа за прорачун борбе

Ова статичка класа садржи четири функције од којих је једна приватна.

`WillHit(int)` приватна функција прима један параметар који представља прецизност. Ова функција прорачуна насумично одабран број од нула до сто и проверава да ли је он већи од прецизности. Ако јесте враћа нетачно а у супротном враћа тачно

`TakeDamage(float, int, int, int, float)` функција прима пет параметара који представљају нанету штету и пробој оклопа непријатељске јединице као и оклоп, одбрану и здравље јединице која ју позива. У овој линији кода:

```
float totalDamage = (damage - (armor * ((1f * armorPiercing) / 100)))
* ((100f - defence) / 100);
```

прорачунава се нанета штета. У случају да је прорачуната штета мања од нула она се поставља на нула. Повратна вредност ове функције је разлика између прослеђеног здравља и прорачунате штете.

`HandleHealth(Image, float, float)` прима слику која представља здравље, тренутно здравље и почетно здравље, и онда поставља попуњеност слике на количник између тренутног и почетног здравља. Ако је тренутно здравље мање или једнако нула ова функција враћа тачно а у супротном нетачно. Повратна вредност ове функције говори јединици која ју позива да ли треба да умре или не, јединица умире ако ова функција врати тачно.

`Attack(float, float, float, IDamageable, UnitStats.Base, bool = false)` функција прима шест параметара тренутно време до следећег напада, раздаљину од непријатеља, време између напада, класу која је задужена за примање штете нападнутог, основне податке о нападачу и изборни параметар да ли је нападач зграда који обично садржи вредност нетачно. Класа која је задужена за примање штете увек имплементира интерфејс *IDamagable*. Прво се проверава да ли мета постоји и ако не постоји враћа се минус два. Потом се проверава да ли је тренутно време до следећег напада мање или једноако нула и да ли је раздаљина мања или једнака домету јединице, ако није враћа се минус један. Ако су ти услови испуњени проверава се да ли је нападач зграда или да ли важи да је раздаљина већа или јендака од 2 и да напдач не ради нула штете на даљину, јер ако је ово тачно нападач ће гађати док ће у супротном морати да се туче. Ако нападач гађа проверава се да ли је он погодио позивом функције `WillHit(int)` ако јесте позива се функција `TakeDamage(float, int)` од нападнутог и прослеђују јој се напад на даљину и пробој на даљину нападача. Ако се нападач туче само се позива функција `TakeDamage(float, int)` нападнутог али се прослеђују напад и пробој оклопа на близину. Када се нападач туче или гађа увек се на крају позива следећа линија кода

`return attackCooldown + attackCooldown * baseStats.shootingSpeed;`
која као повратну вредност функције шаље време до следећег напада.

11 Прављење јединица

Прављење јединица почиње од приказа иконица за јединице које зграда може да прави на корисничком интерфејсу, ово обезбеђује класа *ActionFrame*. У тој класи се обављају све функционалности везане за приказ и интеракцију са дугмићима на корисничком интерфејсу.

`SetActionButtons(PlayerActions, Transform, GameObject = null)` ова функција прима два обавезна и један изборни параметар параметра. Први параметар објекат класе *PlayerActions*, ова класа садржи податке о јединицама и зградама које се могу правити. Други параметар је трансформација тренутно одабраног објекта и она се у овој функцији само памти у промењивој `currentSelection` да би се у другим функцијама прихватали позиви који долазе из компоненти одабраног објекта. Трећи параметар није обавезан и ако није прослеђен поставља се на `null` вредност. Он представља место окупљања за јединице. Прво се проверава да ли први параметар има неку конкретну вредност јер у супротном нема потребе за прављењем дугмића. Ако има вредност сва три параметра се постављају у одговарајуће глобалне промењиве (`actionsList`, `currentSelection` и `rallyPoint`). Потом се проверава да ли постоје јединице које се могу градити у првом параметру. Ако постоје у одабраном објекту се проналази компонента која припада класи *BuildingBuildQueue* и поставља се у глобалну промењиву `buildQueue` која је задужена за ред прављења јединица. Позивом функције `PauseTimer()` из промењиве за ред прављења, зауставља се тајмер прављења јединица. Тајмер се зауставља да се нека од јединица не би направила пре него што се на кориснички интерфејс дода дугме које ју представља у реду прављења. Сада се пролази кроз сваку јединицу која се може правити и за сваку од њих се инстанцира ново дугме за прављење са одговарајућом иконицом и задаје му се име јединице као његово име па се додаје на кориснички интерфејс у део са дугмићима за прављење јединица и додаје се у листу тих дугмића. Свако дугме за прављење има функцију која се позива поводом клика на њега и та функција само директно позива `StartSpawnTimer(string)` класе *ActionFrame* и прослеђује јој своје име. Када се поставе сва дугмад за прављење проверава се да ли у реду прављења већ постоји нека јединица. Ако постоји пролази се кроз све индексе који су сачувани у реду прављења и за сваки се поставља дугме за јединицу у реду прављења. Дугмад за јединице у реду прављења се постављају тако што се позива функција `AddButton(Units.Unit, int, Transform, float = 0)` и њој се прослеђује прва јединица у реду прављења, индекс те јединице, и тренутно одабран објекат. Код позива за

прву јединицу у реду прослеђује се као крајњи параметар и време њеног тајмера. Када се сви дугмићи за јединице поставе тајмер се поново покреће позивом функције `ContinueTimer()`. Када се заврши са јединицама проверава се да ли постоје зграде које се могу градити у првом параметру. Ако постоје поступак је исти као и у случају јединица само што не постоји ред грађења и све што је за њега везано се не мора радити.

`ClearActions(Transform)` функција прима параметар који говори од ког објекта је дошао позив. Ако је тај објекат исти онај који је тренутно одабран ова функција ће уклонити све дугмиће на корисничком интерфејсу. Прво се проверава да ли је прослеђени објекат различит од тренутно одабраног, ако јесте одмах се излази из функције, а ако није наставља се даље. Потом се пролази кроз свако дугме за прављење па кроз свако дугме за ред прављења која се налазе у својим листама и уништавају се. Када су сва дугмад уништена, то јест уклоњена са корисничког интерфејса њихове листе се чисте од свих елемената. На крају се заборавља тренутно одабран објекат из глобалне промењиве да би се игнорисали његови будући позиви функција.

`StartSpawnTimer(string)` функција прима један параметар који представља назив објекта који треба да се прави. Она прво проверава да ли прослеђен назив припада јединици и да ли у глобалној промењивој постоји ред прављења. Ако су оба упита тачна само се позива истоимена функција из реда прављења и прослеђује јој се назив јединице. У супротном се проверава да ли прослеђени назив припада згради. Ако припада онда се проналази која зграда треба да се направи поставља се као одабрана зграда у *BuildingBuildManager* и пали се режим изградње у *PlayerManager*. Ако ни једна од провера није тачна записује се грешка која говори да назив не припада објекту који може да буде направљен и завршава се функција.

`RemoveUnitFromBuildQueue(int)` функција прима индекс јединице у реду прављења која треба да се обрише, и онда само позива истоимену функцију са истим параметром из реда прављења. Једина сврха ове функције је да прими позив из дугмета за ред прављења и проследи га у ред прављења. Сва дугмад за ред прављења при клику позивају ову функцију.

`IsBuilding(string)` и `IsUnit(string)` проверавају да ли је назив објекта који су добили као параметар зграда или јединица респективно. Они пролазе кроз листе зграда и јединица које се могу правити у упоређују прослеђени назив са називом из листи и враћају објекат ако нађу а `null` ако не нађу исти назив.

`AddButton(Units.Unit, int, Transform, float = 0)` функција додаје дугмиће у део корисничког интерфејса за ред прављења за сваку јединицу која се налази у реду прављења. Први параметар представља јединицу која треба да се прави, други њен индекс у реду прављења,

трећи објекат који позива функцију а четврти је обично једнак нули али ако се проследи нека вредност она представља протекло време. Прво се проверава да ли је објекат који је позвао функцију различит од оног који је тренутно одабран, ако јесте излази се из функције док се у супротном наставља даље. Инстанцира се ново дугме коме се као назив додељује његов индекс у реду прављења. Објекат који је прво дете тог дугмета то јест његов први под објекат се поставља за последњи, да би се он приказивао преко осталих јер је он полупровидни зелени слој који представља преостало време тајмера. Тајмер дугмета се поставља на вредност времена потребног за прављење прослеђене јединице. Дугме се додаје на листу дугмади у реду прављења и онда се проверава да ли у тој листи постоји само један елемент. Ако постоји само један то значи да је то први елемент у реду и позива се функција `StartTimer(float = 0)` компоненте класе `BuildQueueAction` којој се прослеђује последњи параметар, па она одузима то прошло време од времена потребног за прављење и покреће свој тајмер.

`RemoveButton(int, Transform = null)` прима индекс дугмета које треба да се уклони и објекат који је позвао функцију. Ако је објекат који позива функцију исти онај који је и одабран дугме се уклања. Потом се проверава да ли постоји још дугмади у реду прављења и ако постоји оном које је сад прво по реду се покреће тајмер.

11.1 Класа реда за прављење

Класа задужена за ред за прављење је *BuildingBuildQueue*. Свака зграда има свој објекат који је инстанца ове класе. Она је задужена за ред прављења те зграде и онда води рачуна о њему чак и када зграда није одабрана.

`StartSpawnTimer(string)` прима назив јединице која треба да се направи и започиње њено прављење ако је постоји довољно ресурса за ту јединицу. Прво се проверава да ли прослеђени назив припада некој од јединица позивом функције `IsUnit(string)` која је иста као и она у *ActionFrame*. Ако припада тип јединице се памти и проверава се да ли постоји довољно ресурса за њено прављење позивом њене функције `TakeResources()` која врати тачно и одузме ресурсе ако има довољно ресурса, а у супротном враћа нетачно. Ако је враћено тачно у листу реда времена прављења `spawnQueue` се убацује време прављења а у листу реда јединица за прављење `spawnOrder` се убацује јединица и њен индекс, који одређује глобални бројач, се убацује у листу индекса. Додаје се дугме позивом функције `AddButton(Units.Unit, int, Transform, float = 0)` из *ActionFrame*. Потом се глобални бројач повећава за један. На крају ако постоји само један елемент у реду прављења позива се функција `StartTimer(float)` од објекта класе *ActionTimer* који се чува у глобалној промењивој и прослеђује му се време прављења те прве

јединице у реду. Та функција намешта потребно време из прослеђене промењиве и започиње одбројавање тајмера.

`SpawnObject()` креира нови објекат на основу *Prefab*-а из прве јединице у реду и додељује у његове компоненте вредности података о тој првој јединици. Потом у хијерархији објеката проналази онај који треба да буде над објекат креираног и поставља га за над објекат. Када се објекат креира позива се уклањање дугмета првог објекта и он се уклања са свих листи. Ако постоји још јединица у реду поново се започиње тајмер са временом од следеће јединице, а ако је ред празан тајмер се зауставља. На крају се пушта звук за завршетак тренинга јединице.

`RemoveUnitFromBuildQueue(int)` прима индекс у реду прављења јединице и уклања је из тог реда. Прво се одреди која по реду је јединица у реду прављења и уклони се дугме из корисничког интерфејса. Ако јединица са прослеђеним индексом није прва по реду онда се њени ресурси враћају и она се уклања са свих листи. Ако јединица јесте прва по реду, тајмер мора да се заустави да се не би случајно направила док је у процесу уклањања. После заустављања тајмера ресурси за ту јединицу се враћају и она се брише исто као и да није прва само што се после проверава да ли је остало јединица у реду и ако јесте започиње се тајмер са временом прве. На крају се пушта звук да је прекинут тренинг.

`PauseTimer()` и `ContinueTimer()` функције само позивају функције за заустављање и настављање тајмера из објекта класе *ActionTimer* који се налази у глобалној промењивој.

Литература

[1] Unity Learn

[2] Lithex Productions, How to BUILD and LAUNCH an RTS Game in UNITY.

[3] Aron Granberg, A* Pathfinding Project