**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: 3D Многопотребителна игра в Unity Engine

Дипломант: Научен ръководител:

*Иво Гергинов               Стоил Стоилов*

СОФИЯ

2017

# ОТЗИВ

# УВОД

В днешно време компютърните игри са основна част от нашето развлечение. Те имат дълга история започваща още през 70-те години на 20-ти век и са претърпели множество подобрения от тогава, било то в качеството на графиките и звука, игровите механики, историите, които разказват и други.

Една от най-важните черти на игрите е възможността двама или повече играчи да играят заедно.  Първоначално това се е случвало по най-простия начин - на една машина (конзола или компютър) с два контролера или споделяне на клавиатурата. По-късно се появява възможността машините да се свързват помежду си в локални мрежи, позволявайки по-големи групи от хора едновременно да играят заедно, и през първите две, три години на 21-ти век се появява възможността за игра през Интернет.  
 Това не само позволява на играчи от различни точки по света да играят на каквито и да било игри заедно, но и поставя началото на MMO (Massive Multiplayer Online) игрите.

Освен в технологиите стоящи зад самите игри, значителни подобрения има и в технологиите за разработка на игри. Използва се всичко от API-та от ниско ниво като DirectX за Windows базирани системи и OpenGL за Windows и Linux до игрови engine-и като Unreal Engine, CryEngine или Unity, които до голяма степен улесняват процеса на разработка на игри като го свеждат до писане на игрови логика и механики посредством API-тата от високо ниво на съответния engine.

Безплатният достъп до тези платформи е позволила масовата поява на малки независими разработчици на игри състоящи се от индивидуални лица или от екипи по няколко души. Това води до много голямо разнообразие в жанровете, разработката на много нови и интересни механики, както и чисто артистични идеи реализирани на базата на вее съществуващи такива.

# ПЪРВА ГЛАВА - ИГРОВИ ENGINE-И. MULTIPLAYER ИРГИ.

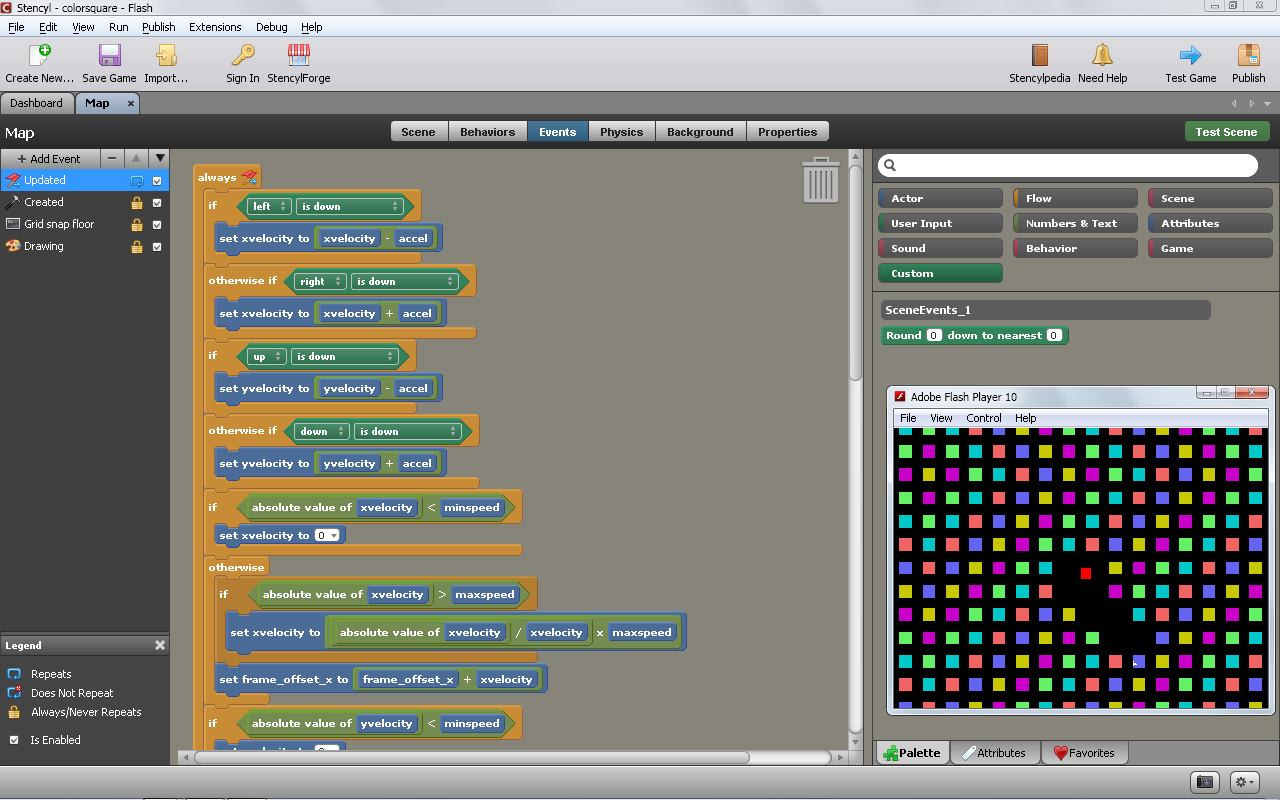
## Игрови Engine-и

Освен огромното развитие на платформите за разработка на игри (игрови engine) от технологична гледна точка, интересно развитие е също и фактът, че голяма част от тях са станали достъпни за всички. В зависимост от платформата достъпът може да е напълно безплатен, или чрез абонамент. Това включва engine-и като Unreal Engine, CryEngine, Unity, GameMaker Studio, Stencyl и други.

Една от основните характеристики на engine-ите e това какъв вид графика поддържа – двуизмерна или триизмерна. Повечето съвременни платформи работят и с двата вида графика, което свежда този въпрос до това на коя платформа най-добре се работи със съответно 2D или 3D. Този избор зависи от жанра на играта, която се разработва. Когато се говори за работа с 2D графика в игрови engine, почти винаги се има предвид работа с вид изображения наречен sprite [спрайт]. Статичните модели в една 2D игра представляват единични sprite-ове, докато анимираните представляват последователност от такива. Всичките изброени по-горе платорми дават възможност да се разработи 2D игра на този принцип, с различна степен на улесненост, но не всички поддържат 3D графика. Това е изцяло поради предназначението им.

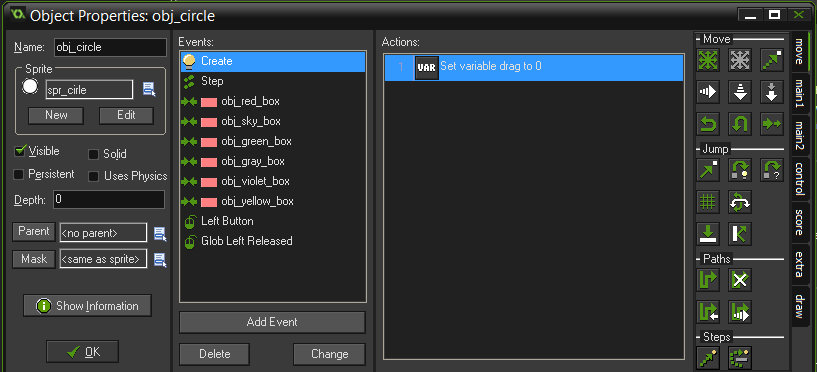
Друга важна черта е колко лесно се използва дадената платформа. Повечето игрови engine-и изискват знания по програмиране до някаква степен. Но в някои от тях разработката на логиката се случва чрез графичен интерфейс, което прави разработката на игри по достъпна до начинаещи и любители. Те дават старт не само в сферата на игрите но и в разработката на софтуер като цяло.

Платфорим като Stencyl и GameMaker Studio се използват предимно за 2D игри. Stencyl е платформа за разработване на 2D игри базирана върху Flash технологията на Adobe. В нея логиката на играта се реализира чрез графичен интерфейс (фиг. 1.1.1.), или чрез писане на скриптове на езика ActionScript, на който е базирана Flash технологията. Тази платформа най-често се използва за разработка на браузърни игри.



### Графична реализация на алгоритъм в Stencyl

На подобен принцип работи GameMaker Studio. В графичния програмен интерфейс на GMS (фиг 1.1.2.) се създават алгоритми с помоща на така наречените програмни библиотеки, които съдържат основни команди, които се използват най-често в игрите като цяло (движение на герой, взимане на вход от играча и други). Ако разработчика иска да създава собствени програмни библиотеки за Game Maker, може да ги пише на специално създадения за engine-a език Game Maker Language. Game Maker Studio поддържа и 3D графика, но е по-популярен в разработката на 2D игри.

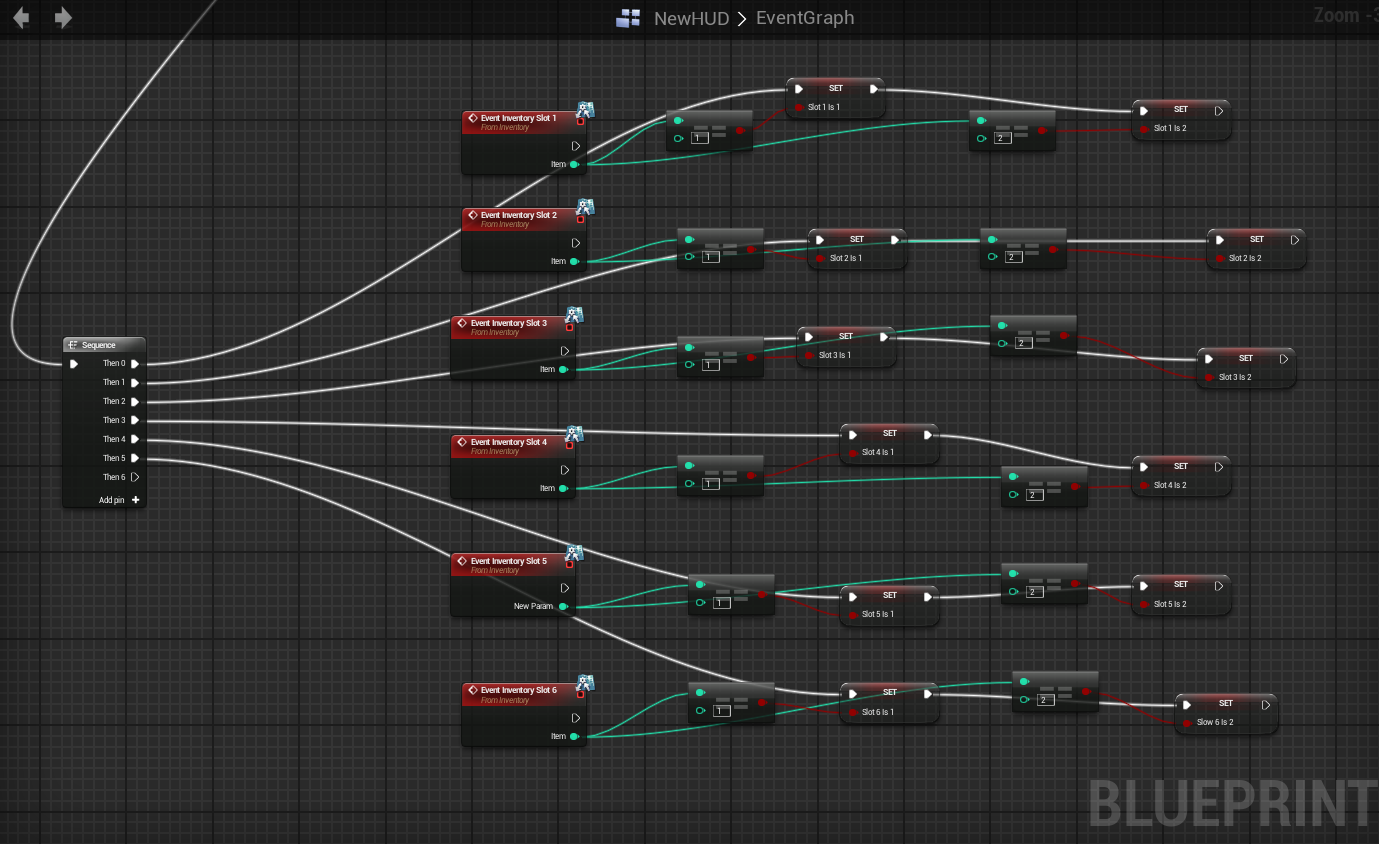


### Графичният интерфейс за програмиране в Game Maker Studio

Unreal Engine (фиг 1.1.4), CryEngine (фиг 1.1.5) и Unity са най-популярните игрови engine-и за разработка на 3D игри. Докато Unity е бил разработен като платформа, която да бъде използвана от външни за компанията Unity Technologies хора, предимно малки независими разработчици, Unreal и CryEngine започват като вътрешни за студиата платформи, на които са се разработвали игри само от самите студиа. И трите платформи изискват някакво ниво на знания по програмиране, но и при тях има предимства за начинаещите в сферата на програмирането и разработката на игри. Те се изразяват в огромни библиотеки от учебни видеа продуцирани от експерти от фирмите разработили съответните платформи, пълни документации и ръководства за скриптовите им API-та, както и множество от асети (от думата asset - актив) изразяващи се във фрагменти код, модели, звукови ефекти, музика и други.

В тези платформи скриптовете за игрите се пишат на конвенционални езици за програмиране като С++, С#, JavaScript и други. Unreal Engine, обаче също разполага с метод за програмиране чрез графичен интерфейс наречен Blueprint (фиг 1.1.3).

Този начин на работа прави тези платформи малко по-трудно достъпни за начинаещи, но за сметка на това възможностите, които предоставят са много по-неограничени. Освен голяма свобода в имплементирането на игрова логика, тези платформи дават възможност за надграждане и модифициране с цел оптимизация на играта или имплементация на някаква нова игрова механика, която би била трудна за имплементиране в стандартните им версии.



### Графично скриптиране в Unreal Engine

## Multiplayer игри

Multiplayer (за много играчи) игрите съществуват откакто съществуват игрите като цяло. Играта Pong, която се счита за първата видео игра в историята, все пак, е била за двама. В първите години на историята на игрите, когато те са се играли най-вече на конзоли, винаги е имало възможност поне двама човека да играят по едно и също време. Това в най-често се случва посредством разделяне на екрана на две или на четири (splitscreen) като всяка от тези части съсответства на един играч. На този принцип работели първите състезателни игри, както и първите shooter-и (от стрелец). Това се превръща в традиция за multiplayer игрите, поради което тази функция присъства в множество игри до ден днешен.

Въпреки популярността на тази функция, splitscreen има един значителен недостатък. Поради разделянето на екрана с цел да се направи място за другите играчи, екранът на всеки отделен играч става прекалено малък. Това не би било проблем при широкоекранните телевизори, които имаме днешно време, но във времето, когато игрите са се играли на малки телевизори и монитори, това е било изключително неудобно. Затова в следващите поколения конзоли, както и игри се имплементира възможността да се играе в мрежа. В началото това означавало няколко машини (конзоли или компютри) да се свържат в локална мрежа и всички играчи да играят заедно в едно помещение. Това също е функционалност, която се използва в днешно време, но и тук има един недостатък. За да се играе в локална мрежа, всички машини, които ще участват в играта трябва да са в едно помещение. Това не би било проблем за група от двама или трима, но за по големи групи (от порядъка на 8 до 10 човека) биха били нужни поне 5 машини, което доста често е изисквало пренасяне на хардуер между помещения (или квартали) и връзване на всички машини в мрежа. С развитието на мрежовите технологии, в първите години на 21.век се появяват първите игри, които поддържат multiplayer през интернет. Това позволило на всеки играч да играе от собствената си машина с всеки друг играч по света.

Multiplayer игрите се различават според жанра, според целта на играта и според това колко играчи са свързани в една игра. Най-популярните жанрове за multiplayer са стратегическите игри, игрите със стреляне от първо или трето лице, ролевите игри, бойните игри и други. Целта на играта може да е кооперативна – малка група от играчи, най-често от двама до четирима, трябва да изпълнят някаква мисия или да проиграят кампанията (историята) на играта заедно. Най-популярния вид multiplayer е състезателният вид, където отделните играчи, или отбори от такива, се съревновават помеждуси. Правилата на това съравнование зависят от жанра на играта и от правилата на режима на игра. Според количеството играчи играта може да е тип арена – от двама до към 16 играча, най-често в отбори.

Добър пример, не само за multiplayer арена, но и за технологичното развитие на multiplayer игрите е поредицата Halo на платформата X-Box. Halo е игра тип first person shooter (стреляне от първо лице), където играчът заема ролята на супервойник от бъдещето. Първата игра от поредицата стана световен хит не само заради грабващата история в соловата си кампания, но и заради забавния multiplayer сегмент. Играчите влизат в арени от всякакъв вид и размер, където се бият помежду си индивидуално или отборно. Връзката се осъществява чрез splitscreen (фиг 1.2.1) или чрез system link – технология за директна локална връзка между конзоли.



### Играта Halo Combat Evolved в режим splitscreen

В следващата част от поредицата вече има възможност за multiplayer през интернет. Играчите се свързват чрез peer to peer система, където един от играчите е домакин на мрежата или чрез специални игрови сървъри, които се поддържат от студиото или от самите играчи, ако те желаят. Връзката със сървърите се осъществява чрез главен (master) сървър, от който всеки игрови клиент получава информация за всеки активен сървър. Тази функционалност се нарича сървърен браузър (фиг 1.2.2). Този модел е много популярен, тъй като позволява на играчите сами да поддържат играта, а и дава възможност за променяне на настройките на игровите режими, редуването на арени и други. Съществува и вариант сървърите да се поддържат изцяло от студиото поддържащо самата игра, в който случай свързването по играчите се случва по други начини (формиране на групи, партии и други).

Играта предоставя голямо разнообразие от режими на игра, както и възможността да се променят правилата на играта за да се създават нови такива. Примери за режими на играта са :

* Slayer – събират се точки като се елиминират други играчи или играчи от друг отбор, ако играта е отборна. Който събере определен брой точки, печели.
* Capture the Flag – на арената има знаме. Това знаме трябва да се вземе и да се занесе в базата на отбора. Играе се до 3 или до 5 взети знамена.
* Оddball – на арената има топка. Един от играчите трябва да задържи топката известно време, като през това време не може да използва оръжията си. Който задържи топката определен брой секунди, печели

и много други. Всички режими имат отборна разновидност.



### Сървърният браузър в Halo 2

Един от по-новите жанрове в multiplayer игрите е MMO – Massive Multiplayer Online (масивен онлайн multiplayer). Този вид игри се състои от един голям постоянно активен свят, в който могат да играят едновременно хиляди хора. В този вид игри би могъл да се инкорпорира почти всеки жанр. Най-популярният жанр за MMO игри е този на ролевите игри.

В подобни игри, ако връзката между играчите не се извършва чрез сървърен браузър, тя се извършва чрез matchmaking система. При нея играчите се свързват чрез присъединяване към плейлист от различни арени и игрови режимове, чрез присъединяване към партия или група, или по ранг, който се определя от статистиките на играча.

Добър пример за такава игра е World of Warcraft (фиг. 1.2.3) на студиото Blizzard Entertainment. В тази игра играчът заема ролята на фантастичен герой, който той създава и следва историята на даден клас герои, който той избира когато създава героя си. Целта на играта е да следва историята, да изпълнява мисии заедно с приятелите си и да изследва големия фантастичен свят, пред който е изправен. Тук игровите сървъри (фиг. 1.2.4) представляват инстанции на един и същ свят, различаващи се по реалното си географско положение, по това дали е възможно играч от единия отбор може директно да влезе в сражение с другия. Също така предоставят информация за това колко населен е сървъра.



### Сървърен браузър в играта World of Warcraft

### Кадър от играта World of Warcraft

# ВТОРА ГЛАВА – ИЗИСКВАНИЯ КЪМ ДИПЛОМНАТА РАБОТА. ИЗБОР НА ТЕХНОЛОГИЯ ЗА РАЗРАБОТКА. АЛГОРИТЪМ НА ИГРАТА.

## Изисквания към дипломната работа

* Да се разработи 3D multiplayer игра в Unity Engine
* Да се разработи dedicated сървър за организиране на онлайн игра
* Да се разработи master сървър за реализиране на сървърен браузър и лобии
* Да се имплементира комуникацията между клиент и сървър

## Избор на технология за разработка

За разработката на играта е избран игровия engine Unity, поради най-голяма запознатост с неговото API. В Unity скриптовете се пишат на езиците за програмиране C# и JavaScript. Избраният език за писане на скриптовете е C#, поради по-голям опит с него, което поражда предпочитание към него.

### Основни понятия и класове в Unity Engine

Това, което прави Unity лесен за използване е огромното наличие на готови ресурси наречени асети, които начинаещите разработчици могат да използват като база за развитие. Тези асети най-често са организирани в контейнери наречени префабове, които съдържат модели и/или скриптове.

API-то на Unity е много обширно и е малко вероятно в една игра да бъде нужна употребата на всеки един клас в него, но има няколко основни класа, които стоят в основата на скриптовете в Unity:

* MonoBehaviour – базовият клас от който наследяват всички скриптове в Unity. В него се съдържат основните методи в които се изпълняват всички скриптове – Awake(), Start(), Update(), FixedUpdate() и други. Awake() се извиква веднъж когато инстанция на даден скрипт се зареди в сцената. Start() се извиква веднъж след като скрипта бъде активиран. Този метод попринцип се използва за иницализиране на променливи. Update() се извиква веднъж на всеки кадър. Колко често ще се извиква този метод зависи от скоростта на обновяване на кадрите (framerate). FixedUpdate() е подобен на Update() с тази разлика, че се извиква на фиксиран интервал независим от скоростта на обновяване на кадъра. В този метод е най-удачно да се изпълнява код свързан със системата за физика на Unity.
* NetworkBehaviour – Наследява MonoBehaviour и позволява употребата на мрежовото API на Unity. Всички скриптове свързани с multiplayer частта на играта трябва да наследяват NetworkBehaviour
* Input – Класът съдържащ всичката функционалност свързана с четене на вход от играча.
* NetworkManager – Помощен клас при разработката на multiplayer игри в Unity. Позволява надграждането на NetworkManager компонента, който Unity предлага като база. NetworkManager компонента управлява състоянието на дадена сцена, която е в мрежа: връзка между клиенти и сървъри, обновяване на данните за другите играчи и обекти в сцената.
* GameObject – базовият клас на всички обекти в дадена Unity сцена. Използва се като контейнер за всякакви ресурси, които присъстват в сцената - модели, изображения, данни и др. Всеки един скрипт представлява компонент на GameObject-а и по този начин се изпълнява неговата функционалност в дадена сцена.

## Основен алгоритъм на играта

Играта представлява стреляне от трето лице с top-down изглед (камерата на играча стои над неговия герой). От към правила и механики е силно опростена. Играчите са поставени в арена. Целта е да се събере даден брой точки като се убиват другите играчи. В играта има система за персонализиране на бронята на героя на играча.

Multiplayer функционалността е имплементирана чрез NetworkManager компонента на Unity, както и модифицирана версия на примерния Unity Master Server проект. Мрежовият сегмент на играта следва dedicated server модела, а master server-а служи за имплементация на сървърен браузър. Модификациите се състоят в добавяне на функционалност за автоматично отваряне на порт за връзка извън локална мрежа, тъй като базовата имплементация работи само в локална мрежа. Това е осъществено чрез Mono.Nat библиотеката. Тази библиотека също дава достъп до публичното IP на сървъра, което е ключово за връзката между клиента и сървъра.

С цел играта да има възможност да бъде поддържана от потребителската и база, игровият клиент, игровият сървър и Master-Server-a са разделени на 3 отделни един от друг компонента. Master-Server-a е компилиран отделно от останалите сцени, а игровите клиент и сървър са разделени посредством потребителския интерфейс. За разлика от този на клиента, сървърския интерфейс позволява единствено да се регистрира и да се пусне сървър. Това се прави с цел даден играч да не попадне в притежание на две копия на играта. Крайния пакет на играта ще включва самата игра, т.е. игровия клиент и един сървър, който може да бъде пуснат на машина по избор или да бъде хостнат на външен сървър, това зависи изцяло от потребителя.

Играта като цяло е разделена общо на 3 сцени. Първата сцена е тази съдържаща главното меню и системата за персонализиране на бронята на героя. Втората сцена представлява основната игрова сцена, където в зависимост дали компилационната единица е игрови клиент или сървър на потребителя е представен съответният потребителски интерфейс. В игровия клиент са компилирани и двете сцени, докато в сървъра – само втората. Третата сцена представлява Master-Server-a.

### **Игровият клие**нт

Когато игровият клиент се свърже с master-server-a, той изпраща NetworkMessage на клиента съдържащ списък с активните игрови сървъри. В този списък се съдържат имената, IP адресите, както и портовете на игровите сървъри. Когато клиента получи списъка го извежда върху потребителския интерфейс под формата на бутони. Когато играчът избере някой от сървърите бива прехвърлен на интерфейса на лобито, където вижда списък с останалите играчи и дали те са готови за игра. Играчът стои свързан с Master-Server-a само докато взима списъка със сървъри от него. Това се случва в началото, когато се зареди екрана за списъка със сървъри или ако играча избере опцията за обновяване на списъка.

Когато когато играчите са готови, сигнализират на сървъра да започне играта чрез Command метод, което зарежда настоящата арена. Скрипта на арената зарежда оръжията за съответната карта на съответните им места и играчите биват spawn-нати на арената. Щом това стане играчите започват да играят. В PlayerController класа се обработва входа на играча, както и неговото взаимодействие с оръжието, което държи. Всички видове оръжия наследяват от класа CombatItem, който съдържа основни данни за тях, като например какви щети нанася съответното оръжие. В този класи има и два виртуални метода Action1() и Action2(), които позволяват на играча да контролира оръжието. Например ако оръжието е огнестрелно, Action1() ще стреля, а Action2() ще презарежда.

В класовете наследяващи CombatItem се имплементира функционалността на съответния вид оръжие. Например класа GunController имплементира основната функционалност на всички огнестрелни оръжия. Когато играча използва GunController, това инстанцира обект от тип Bullet, чиято функционалност е да се транслира напред в рамките на определено разстояние. Скороостта и разстоянието са сериализирани параметри, които се променят според вида куршум. В класа GunController и класа Bullet() се съдържа препратка към играча с цел да се отброяват точките на играча. Когато обект от тип Bullet влезне в колизия с играч, той губи точки живот. При колизията Bullet обекта проверява дали уцеления играч е на 0 точки живот и ако е така, казва на PlayerController обекта да увеличи точките на играча с 1. Когато някой играч достигне нужния брой точки, изпраща Command метод към сървъра за да спре играта и да прехвърли играчите към лоби екрана, където се виждат резултатите им. В тозим момент играча избира дали да продължи, или да излезне от този сървър.

### Игровият сървър

Когато игровият сървър бъде стартиран, той се регистрира в списъка на master-server-a с NetworkMessage съдържащ неговия IP адрес, порт, име и максимален брой играчи. Докато е активен чака команди за започване на игра, след което извършва всички функции, които извършва сървърната страна на NetworkManager компонента повреме на играта, докато не получи команда за край на играта. Този процес се извършва циклично докато сървърът работи. Ако сървърът бъде спрян, се дерегистрира от списъка със сървъри.

### Master-Server

Всички описани по-горе процедури свързани с Master-Server-a се извършват в скрипт наречен Master-Client. Самият сървър единствено слуша за връзка от игрови клиент или сървър и изпълнява съответната процедура от своята страна. Това включва записване и махане на сървъри от списъка, както и изпращане на списъка към свързалите се клиенти.

### Персонализиране на героя

Персонализирането на героя се състои от изглед към героя и интерфейс за избор на частите на бронята. Разделени са на 3 раздела – части за раменете, за тялото и за краката. Текущо избраните части на бронята на играча стоят върху модела като деца на GameObject-а на играча. Когато бъде избрана дадена част от бронята, нейното име се добавя като запис в PlayerPrefs речника.Добра практика би била частите да се зареждат и изчистват от сцената в зависимост от това в кой раздел се намира играча. По този начин ако количеството налични части е прекалено голямо, не се претоварва паметта, а е възможно да се увеличи с течение на времето и обновяване на съдържанието на играта. При настоящето количество части това обаче не е нужно. Безпроблемно могат да бъдат заредени абсолютно всички части в сцената, да бъдат инстанцирани и съответната инстанция на частта да се активира или деактивира в зависимост от това дали е избрана или не. Избраната част се записва в PlayerPrefs веднага щом бъде избрана, затова няма нужда играча извънредно да запазва настройки, когато излиза от интерфейса за персонализиране на героя.

# ТРЕТА ГЛАВА - ОПИСАНИЕ НА КОДА НА ИГРАТА

## Игровият клиент

### Класът PlayerController

public class PlayerController : NetworkBehaviour

{

[SerializeField]

// The movement speed of the player

float Speed;

[SerializeField]

//Reference to the player camera

Camera PlayerCam;

[SerializeField]

// The transform of the gun origin

// That is the position, where the gun will be in relation to the player

Transform gunOrigin;

//Checks whether the player is using a gamepad controller or not

bool OnGamepad;

//Vector3 variable used in the movement method

Vector3 MoveData = Vector3.zero;

//The player's RidgidBody component

//This is used to apply physics

Rigidbody PlayerRB;

//The weapon the Player is currently using

CombatItem Weapon;

Това е класът с основната функционалност на играча. Тук се намират всички променливи и методи свързани с контролирането на играча.

В Start() метода на PlayerController се инициализират RidgidBody променливата, която е нужна за движението на играча, както и камерата на играча, в която се съдържа CameraControl скрипта, който и позволява да следва позицията на играча по X и Z осите. Методите Move(), Rotate() и GamepadRotate() са стандартни методи за придвижване на героя. Всички методи четат вход от играча посредством Input.GetAxisRaw() метода, който следи за вход от контроли описани в Input Manager системата на играта. Move() метода придвижва играча посредством Ridgidbody.MovePosition метода. Въртенето на играча се случва по два начина в засисимост от контролния метод. Ако се използва геймпад контролер, например X-Box контролер, се преизчислява посоката на според входа от джойстика на контролера. Накъдето е насочен джойстика, натам ще сочи играча.

void GamepadRotate()

{

Vector3 direction = Vector3.right \* Input.GetAxisRaw("GPLookHorizontal")

+ Vector3.forward \* -Input.GetAxisRaw("GPLookVertical");

if(direction.sqrMagnitude > 0.0f)

{

transform.rotation = Quaternion.LookRotation(direction, Vector3.up);

}

}

Ако се използва мишка, се проектира Raycast лъч от камерата на играча до пода на арената, като този лъч минава през позицията на курсора върху екрана. От този лъч се взима точката, в която пробожда равнината на пода и играча се завърта чрез Qaternion.LookRotation метода към тази точка само по X и Z координати.

void Rotate()

{

float rayL;

Ray camRay = PlayerCam.ScreenPointToRay(Input.mousePosition);

Plane floor = new Plane(Vector3.up, Vector3.zero);

if (floor.Raycast(camRay, out rayL))

{

Vector3 lookPoint = camRay.GetPoint(rayL);

transform.LookAt(new Vector3(lookPoint.x, transform.position.y, lookPoint.z));

}

}

CheckConrolMethod метода следи дали се използва контролер или клавиатура и мишка като контролен метод. Ако се използва контролер, ротацията се извършва посредством GamepadRotate, a ако се използва мишка, се ротира посредством Rotate метода.

PickUp позволява да вдигне оръжие от земята. Ако играчът влзене в колизия обект с таг „CombatItem“, има опцията да вдигне оръжието, което изпълнява този метод. При вдигане на оръжието, то бива поставено в позицията на GunOrigin Transform-a, става child обект на GameObject-a на играча и в Weapon променливата се запазва препратка към CombatItem компонента на оръжието.

void PickUp(GameObject g)

{

g.transform.parent = transform;

g.transform.position = gunOrigin.transform.position;

g.transform.rotation = gunOrigin.transform.rotation;

g.SetActive(true);

g.GetComponent<GunController>().enabled = true;

CombatItem c = g.GetComponent<CombatItem>();

if (Weapon != null) Drop(Weapon.gameObject);

c.SetEquiped(true);

Weapon = c;

}

Методите Action1Input и CmdAction1 са свързани с използването на дадено оръжие. Action1Input просто чете вход за Action1 на оръжието, което играча държи в момента. CmdAction1 изпраща метод с Command атрибут на сървъра, който се изпълнява върху него. В такива методи най-често има мрежово инстанциране (spawn-ване) на обекти в мрежова сцена по инициатива на играча. В случайя това е така, защото имаме spawn-ване на обект от тип Bullet (куршум).

### Класът CameraControl

Този скрипт съдържа метод, който позволява на камерата на играча да следва неговия GameObject

### Класът OnlineSetup

Тук се съдържа скрипт за деактивиране на всички камери на играчите освен тази на локалния за съответния клиент играч.

### Класът CombatItem

Това е базовият клас на оръжията в играта. Състои се от булева, която следи дали оръжието се използва или не. Това е важно, тъй като ако се използва, то то е child обект на играча, който има NetworkTransform. Тъй като не може в един обект да има NetworkTransform, и същевременно в негов child също да има такъв компонент, е важно да се знае дали оръжието се използва за да може този компонент да бъде деактивиран докато оръжието се използва. Също така има два виртуален метод Action1, в който се имплементира Action1 функционалността на съответния вид оръжие, когато се създаде клас наслеяващ CombatItem.

public class CombatItem : NetworkBehaviour

{

protected bool equiped = false;

public virtual void Action1(bool on){}

public virtual void Action2(bool on){}

protected virtual void Fire(){}

public bool GetEquiped()

{

return equiped;

}

public void SetEquiped(bool e)

{

equiped = e;

}

}

### Класът GunController

Този клас наследява CombatItem и имплементира функционалност на огнестрелно оръжие. Има следните член променливи: Origin – Transform обект, който показва къде ще се инстанцира куршум при стрелба, Bullet – препратка към Bullet префаба на съответното оръжие. Този префаб съдържа модела на куршума и неговия Bullet компонент. Във FixedUpdate метода на този клас се отброява времето между истрелите на дадено оръжие. В случая оръжието е автоматично, така че когато контрола за стрелба се задържи, на всяко отброяване се инстанцира куршум. Куршум се инстанцира в сцената и в мрежата чрез Fire метода, който се управлява от потребителския вход и брояча във FixedUpdate посредством булевите IsFiring и CanFire. Fire метода се извиква в Action1 метода на GunController класа, като в този метод се управлява булевата IsFiring. CanFire се управлява във FixedUpdate.

### 

void FixedUpdate()

{

ROFCountDown -= Time.deltaTime;

if (IsFiring)

{

if (ROFCountDown <= 0)

{

ROFCountDown = TimeBetweenShots;

CanFire = true;

}

else

{

CanFire = false;

}

}

if (equiped)

{

this.GetComponent<NetworkTransform>().enabled = false;

}

else this.GetComponent<NetworkTransform>().enabled = true;

}

protected override void Fire()

{

if (IsFiring && CanFire)

{

GameObject projectile = Instantiate(Bullet, Origin.position, Origin.rotation) as GameObject;

NetworkServer.Spawn(projectile);

}

}

public override void Action1(bool on)

{

IsFiring = on;

Fire();

}

public override void Action2(bool on)

{

}

### Класът Bullet

Тук е имплементирана функционалността на куршум. Щом бъде инстанцран, куршума започва да се транслира в посока Vector3.forward, което е предната ориентация на обекта. Това се случва във FixedUpdate метода.

void FixedUpdate()

{

transform.Translate(Vector3.forward \* Speed \* Time.deltaTime);

if (Vector3.Distance(transform.position, InitPosition) > Range)

{

Destroy(this.gameObject);

}

}

Куршума се самоунищожава когато стигне до определено разстояние, което е дефинирано в Range променливата. InitPosition се дефинира в Start метода като настоящата позиция на куршума в този момент.

В класа на куршума се съдържа Damage променливата, която определя каква щета ще нанесе оръжието.

Този клас също има препратка към PlayerStats компонента на играча, за да може да му увеличи броя на точките, ако съответния куршум елиминира уцеления играч. Това се случва в OnCollisionEnter метода.

void OnCollisionEnter(Collision c)

{

if(c.gameObject.tag == "Player(Clone)")

{

PlayerStats stats = c.gameObject.GetComponent<PlayerStats>();

stats.ReduceHealth(Damage);

if(stats.GetHealth() <= 0)

{

player.AddKill();

}

}

}

### Класът PlayerStats

Този клас се занимава с игровите статистики на играча повреме на игра. Това са Health, Kills и Deaths. Методите в този клас предимно и най вече се извикват от Bullet обекта, който влиза в колизия с играча. В неговия FixedUpdate се следи дали Health-a на играча е над 0. Ако е, играча умира, неговата Deaths променлива се инкрементира и той се spawn-ва наново.

### Класът MyNetMngr

Надграждане на NetworkManager системата. Тук единствено е добавено автоматично отваряне на портове посредством Mono.Nat библиотеката, което позволява на всички мрежови компоненти да се свързват през интернет. Отварянето на портовете се постига посредством методите OpenPort и DeviceFound, който се имплементира по изискване на Mono.Nat библиотеката.

//Discovers active networking device (router)

// and Calls the DeviceFound method

void OpenPort()

{

NatUtility.StartDiscovery();

NatUtility.DeviceFound += DeviceFound;

}

//Implements the port forwarding procedure

void DeviceFound(object sender, DeviceEventArgs args)

{

INatDevice device = args.Device;

if(device.GetSpecificMapping(Protocol.Udp, 7777).PublicPort == -1)

{

device.CreatePortMap(new Mapping(Protocol.Udp, 7777, 7777));

}

}

Освен, че присъстват в надграждането на NetworkManager компонента, тези методи представляват основната модификация, която е направена върху Master-Server-а предоставен от Unity. Освен това в скриптовете на Master-Server-а е добавена функционалност за взимане на публичното IP, на който той работи. Това е ключово, тъй като мрежовото API на Unity изисква публично IP за връзка установяване на връзка.ЧЕТВЪРТА ГЛАВА – РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛА

## **Инсталация**

Играта, както и сървърът се инсталират от съответно battlegon.exe и battlegonserver.exe. Състои се от стандартна Windows инсталация, потребителя избира инсталационна директория, избира дали иска пряк път на работния плот и завършва инсталацията.

## **Игровият клиент**

Когато играта стартира се извежда главното меню (фиг 4.1.1) на играта. На него се вижда героя на играча, както опциите на играта. От тук играчът може да влезне в сървърния браузър (фиг 4.1.2), откъдето може да се присъедини към игра, да влезне в системата за персонализиране на героя, или да излезне от играта.



### Главното меню на играта

В менюто за персонализация на героя (4.1.3) се избира каква броня да носи героя на играча. Разполага с опции за избор на шлем, броня за раменете и за краката, за раменете и краката ляво и дясно се избират поотделно.



### Меню за персонализация на героя

Когато играча е в сървърния браузър има възможност да избере игрови сървър, към който да се свърже. Щом това се случи, се зарежда директно в арената на съответния сървър.



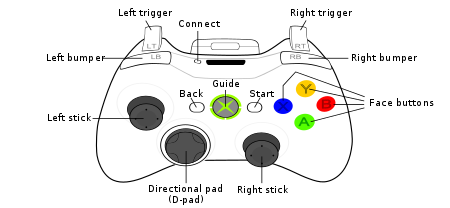
### Сървърният браузър на играта



### Екран от играта

Контролните схеми на играта са следните :

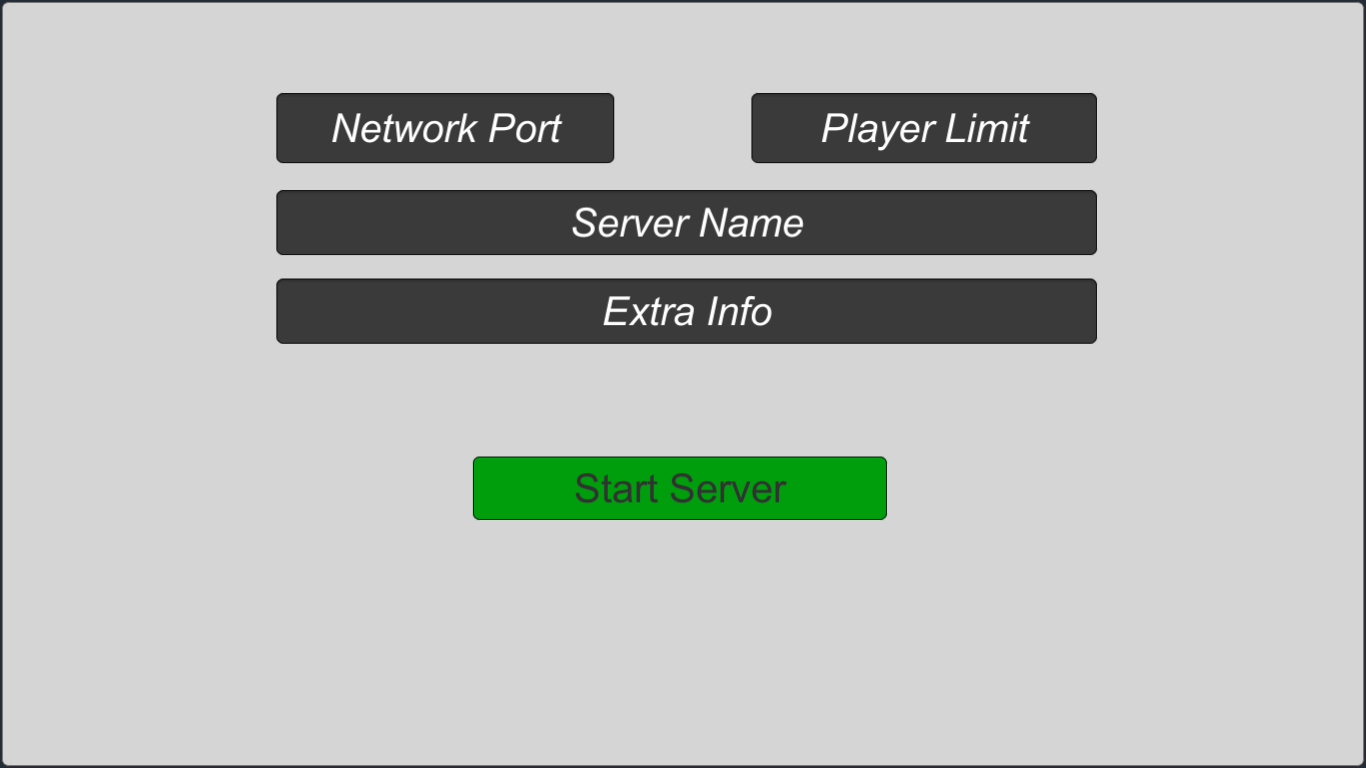
* Клавиатура и мишка: клавишите W, A, S, D или стрелките управляват движението. Въртенето на героя се прави с движение на курсора около играча. Играча винаги е насочен към положението на курсота. Стреля се с ляв бутон на мишката. Оръжие се оборудва с клавиша E.
* Контролер(според фиг 4.1.4): Левият джойстик управлява движението – посоката в която бъде натиснат джойстика е посоката, в която ще се движи героя. Десният спусък стреля. X бутона оборудва оръжието.



### Схема на X-Box 360 контролер

# Игровият Сървър

Игровият сървър представлява един екран, в който се въвежда основната информация за сървъра – мрежовия порт, на който ще върви, максималният брой играчи, името на сървъра и допълнително поле за коментари. Щом бъде въведена нужната информация се натиска зеления бутон Start Server. Поради технически проблеми с настоящата версия е силно препоръчително екрана на сървъра да не се минимизира.



### Игровият сървър

# ЗАКЛЮЧЕНИЕ

Unity е качествен игрови engine и в по-голямата си част лесно достъпен. От друга страна обаче, мрежовата платформа на Unity е по-трудна за усвояване от останалата част на API-то. Unity предлага готови решения, но поради масовата им употреба от множество по-малки игри, услугата Unity Matchmaking не се препоръчва при разработката на по-голям проект. Това е така поради една основна причина: При употребата на Unity Matchmaking играта е ограничена до 20 едновременни потребителя без нуждата от допълнително заплащане. Това не би било проблем за по-успешен проект, но подобно ограничение не е полезно за малки проекти, както и за самото тестване на играта при по-голям мащаб на мрежовата инфраструктура.

Въпреки техническите проблеми с играта към днешна дата, има добра база за развитие. Освен изглаждането на проблемите с мрежовата част, може допълнително да се развие системата на оръжията, да се въведат повече режими за игра, да се разработи ranking система за matchmaking базиран на умението на играча, и други.

За ново-въвеждащите се в разработката на игри с Unity е препоръчително да се захванат с игра за един играч, вместо с multiplayer игра като първи проект.

# ИЗПОЛЗВАНА ЛИТЕРАТУРА

Unity Documentation - <https://docs.unity3d.com/Manual/index.html>

Unity Forums - <https://forum.unity3d.com/>

Unity Answers - <https://answers.unity3d.com/>

Mono.Nat Port Forwarding in C# - <http://www.fluxbytes.com/csharp/upnp-port-forwarding-the-easy-way/>

SQLite in C# - <http://blog.tigrangasparian.com/2012/02/09/getting-started-with-sqlite-in-c-part-one/>

Съдържание

[ОТЗИВ 2](#_Toc482558498)

[УВОД 3](#_Toc482558499)

[ПЪРВА ГЛАВА - ИГРОВИ ENGINE-И. MULTIPLAYER ИРГИ. 5](#_Toc482558500)

[1.1. Игрови Engine-и 5](#_Toc482558501)

[1.1.1. Графична реализация на алгоритъм в Stencyl 7](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558502)

[1.1.2. Графичният интерфейс за програмиране в Game Maker Studio 8](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558503)

[1.1.3. Графично скриптиране в Unreal Engine 10](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558504)

[1.2. Multiplayer игри 11](#_Toc482558505)

[1.2.1 Играта Halo Combat Evolved в режим splitscreen 13](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558506)

[1.2.2 Сървърният браузър в Halo 2 15](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558507)

[1.2.3 Сървърен браузър в играта World of Warcraft 17](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558508)

[1.2.4 Кадър от играта World of Warcraft 17](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558509)

[ВТОРА ГЛАВА – ИЗИСКВАНИЯ КЪМ ДИПЛОМНАТА РАБОТА. ИЗБОР НА ТЕХНОЛОГИЯ ЗА РАЗРАБОТКА. АЛГОРИТЪМ НА ИГРАТА. 18](#_Toc482558510)

[2.1 Изисквания към дипломната работа 18](#_Toc482558511)

[2.2 Избор на технология за разработка 18](#_Toc482558512)

[2.2.1 Основни понятия и класове в Unity Engine 18](#_Toc482558513)

[2.3 Основен алгоритъм на играта 20](#_Toc482558514)

[2.3.1 Игровият клиент 22](#_Toc482558515)

[2.3.2 Игровият сървър 24](#_Toc482558516)

[2.3.3 Master-Server 24](#_Toc482558517)

[2.3.4 Персонализиране на героя 24](#_Toc482558518)

[ТРЕТА ГЛАВА - ОПИСАНИЕ НА КОДА НА ИГРАТА 26](#_Toc482558519)

[3.1. Игровият клиент 26](#_Toc482558520)

[3.1.1. Класът PlayerController 26](#_Toc482558521)

[3.1.2. Класът CameraControl 29](#_Toc482558522)

[3.1.3. Класът OnlineSetup 29](#_Toc482558523)

[3.1.4. Класът CombatItem 29](#_Toc482558524)

[3.1.5. Класът GunController 30](#_Toc482558525)

[31](#_Toc482558526)

[3.1.6. Класът Bullet 32](#_Toc482558527)

[3.1.7. Класът PlayerStats 33](#_Toc482558528)

[3.1.8. Класът MyNetMngr 33](#_Toc482558529)

[4.1. Инсталация 35](#_Toc482558530)

[4.2. Игровият клиент 35](#_Toc482558531)

[4.2.1. Главното меню на играта 35](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558532)

[4.2.2. Меню за персонализация на героя 36](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558533)

[4.2.3. Сървърният браузър на играта 37](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558534)

[4.2.4. Екран от играта 37](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558535)

[4.2.5. Схема на X-Box 360 контролер 38](file:///C:\Users\Ivo%20Gerginov\Documents\Дипломна.docx#_Toc482558536)

[4.3. Игровият Сървър 39](#_Toc482558537)

[ЗАКЛЮЧЕНИЕ 40](#_Toc482558538)

[ИЗПОЛЗВАНА ЛИТЕРАТУРА 41](#_Toc482558539)