Универзитет у Београду

Електротехнички факултет



Имплементација пуцачке игрице 2D видео игре у Godot развојном оквиру

Дипломски рад

|  |  |
| --- | --- |
| Ментор: | Кандидат: |
| Проф. др Марко Мишић | Влада Бељански 2017/415 |

Београд, Септембар 2024.

Садржај

[Садржај 2](#_Toc442888948)

[1. Увод 4](#_Toc442888949)

[2. Припрема мастер тезе 5](#_Toc442888950)

[3. Стил писања текста 6](#_Toc442888951)

[3.1. Организација текста тезе 6](#_Toc442888952)

[3.2. Стил писања, скраћенице, преводи са енглеског 6](#_Toc442888953)

[3.3. Слике, табеле, променљиве у тексту 6](#_Toc442888954)

[3.4. Мерне јединице 6](#_Toc442888955)

[3.5. Програмски код 6](#_Toc442888956)

[3.6. Референце 6](#_Toc442888957)

[3.7. Списак скраћеница, слика, табела 6](#_Toc442888958)

[3.8. Захвалница 6](#_Toc442888959)

[4. Форматирање текста 6](#_Toc442888960)

[4.1. Форматирање текста и стилови у алату *Microsoft Word* 6](#_Toc442888961)

[4.2. Форматирање насловне стране 6](#_Toc442888962)

[4.3. Форматирање текста тезе 6](#_Toc442888963)

[4.3.1. Хијерархија наслова текстуалних целина 6](#_Toc442888964)

[4.3.2. Форматирање текста у оквиру текстуалних целина 6](#_Toc442888965)

[4.3.3. Слике и табеле 6](#_Toc442888966)

[4.3.4. Једначине 6](#_Toc442888967)

[4.3.5. Променљиве 6](#_Toc442888968)

[4.3.6. Набрајање 6](#_Toc442888969)

[4.4. Листа референци (списак литературе) 6](#_Toc442888970)

[4.5. Списак скраћеница, слика и табела 6](#_Toc442888971)

[4.6. Прилози 6](#_Toc442888972)

[4.7. Садржај 6](#_Toc442888973)

[Литература 6](#_Toc442888974)

[Списак скраћеница 6](#_Toc442888975)

[Списак слика 6](#_Toc442888976)

[Списак табела 6](#_Toc442888977)

[A. Подешавање стилова текста за *Word* 2003 6](#_Toc442888978)

[A.1. Подешавање изгледа странице 6](#_Toc442888979)

[A.2. Форматирање основног текста и наслова хијерархијски уређених текстуалних целина 6](#_Toc442888980)

[A.2.1. Основни текст 6](#_Toc442888981)

[A.2.2. Наслов поглавља 6](#_Toc442888982)

[A.2.3. Наслов потпоглавља 6](#_Toc442888983)

[A.2.4. Наслов одељка 6](#_Toc442888984)

[A.2.5. Наслов пододељка 6](#_Toc442888985)

[A.3. Слике 6](#_Toc442888986)

[A.3.1. Позиционирање слике 6](#_Toc442888987)

[A.3.2. Назив и нумерација слике 6](#_Toc442888988)

[A.4. Табеле 6](#_Toc442888989)

[A.4.1. Позиционирање и изглед табеле 6](#_Toc442888990)

[A.4.2. Назив и нумерација табеле 6](#_Toc442888991)

[A.5. Једначине 6](#_Toc442888992)

[A.6. Набрајање 6](#_Toc442888993)

[A.7. Садржај 6](#_Toc442888994)

[A.8. Списак литературе 6](#_Toc442888995)

[A.9. Форматирање наслова за садржај, списак литературе (скраћеница, слика, табела) 6](#_Toc442888996)

[A.10. Наслови хијерархијски уређених целина у прилозима централном тексту тезе 6](#_Toc442888997)

[A.10.1. Први ниво наслова у прилозима 6](#_Toc442888998)

[A.10.2. Други ниво наслова у прилозима 6](#_Toc442888999)

[A.10.3. Трећи ниво наслова у прилозима 6](#_Toc442889000)

[A.10.4. Четврти ниво наслова у прилозима 6](#_Toc442889001)

1. Увод

Овај рад се бави имплементацијом пуцачке 2D видео игре у Godot развојном оквиру. Циљ овог документа је да прикаже комплетан развој ове видео игре, од описа жанра и почетних захтева, преко избора технологије, до саме реализације и изазова који су се јавили током реализације.

У почетку, рад пружа преглед самог жанра и сагледава постојећа решења, односно игара које припадају описаном жанру. Затим, рад уводи читаоца у основне концепте и алате коришћене током развоја игре у Godot оквиру. У наставку се детаљно описују специфичности имплеметације, односно са конкретна решења проблема који су се јавили током развоја овог типа игре користећи изабрану технологију.

Имплементација игре обухвата креирање играча, непријатеља, пројектила и образаца кретања, као и интеракције између тих елемената. Поред тога, имплементиран је и стандардни мени игре. Систем је дизајниран тако да омогућава лако додавање нових функционалности, као што су различити типови непријатеља и пројектила, уз модуларни приступ који подржава лаке измене.

Завшни делови рада посвећени су приказу коначне функционалности игре кроз детаљно корисничко упутство које објашњава рад система. На крају, закључак нуди кратку рекапитулацију целокупног процеса развоја, са освртом на потенцијалне надоградње и могућности унапређења система.

1. Опис проблема

У овом поглављу биће дат преглед жанра, техничких изазова, као и кратка функционална спецификација.

* 1. Преглед жанра

Игре из жанра *vertically scrolling shoot ‘em up*, односно пуцачке игре са вертикалним скроловањем, карактеришу непријатељи који се, углавном, појављују на врху екрана и крећу се према доле, док играч контролише кретање свог лика у четири смера. Главни циљ играча је да избегава непријатељске пројектиле, уништи што већи број непријатеља испаљујући своје пројектиле и преживи до краја нивоа.

Овај тип игре обично садржи таласе непријатеља са различитим обрасцима кретања и различитим врстама пројектила. Пројектили и интеракције између играча и непријатеља чине кључну механику ове игре. Игра може да укључује и систем бодовања који награђује играча за ефикасност у уништењу непријатеља.

Главне карактеристике жанра укључују:

* Генерисање таласа непријатеља са различитим обрасцима кретања.
* Кретање играча у сва четири смера и испаљивање пројектила како би се уништили непријатељи.
* Фокус на избегавању непријатељских пројектила и награђивање играча кроз ефикасно уништавање непријатеља.
  1. Технички изазови и функционалне спецификације

Током развоја игре кључни изазов је био направити систем који је модуларан, проширив и лако подесив, чиме се омогућава лако додавање нових непријатеља, пројектила или чак механика, уз једноставно итеративно тестирање нивоа или нових елемената.

Био је важан дизајн флексибилног система који омогућава креирање различитих типова непријатеља са специфичним обрасцима кретања. Систем је морао подржавати лако подешавање времена појављивања и путања непријатеља.

Поред тога, модуларност система је омогућила лако додавање непријатеља, пројектила и механика, при чему сваки елемент функционише независно, али у јасној интеракцији са другим елементима. Овај приступ, осим што је олакшао развој, омогућио је правилно функционисање интеракција и механика након што су тестиране, јер је једном тестирани систем функционисао исправно у свим деловима игре, без потребе за поновним тестирањем сваки пут када се додају нови елементи.

1. Опис коришћених технологија

Током развоја ове 2D пуцачке игре, главна платформа био је *Godot Engine* који је пружио потребне алате за имплементацију механика игре. Осим тога коришћени су и додатни алати за подршку развоја.

* 1. *Godot engine*

*Godot engine* био је централна технологија у развоју ове игре. То је *open-source* развојни окрвир који подржава развој 2D и 3D игара. У оквиру овог одељка детаљније ће бити објашњени кључни елементи који су коришћени у развоју игре.

* + 1. Чворови и сцене

У оквиру *Godot* основни градивни блокови су чворови( *nodes* ). Чворови су вишенаменски објекти који могу да обављају разне функције, као што су контрола кретања, анимација, обрада уноса итд. Сви чворови организовани су у сцене( *scenes* ), при чему свака сцена може да садржи више чворова или чак друге сцене као под-сцене. Сцена је такође наследник одређеног чвора у објектно оријентисаном смислу, што омогућава хијерархијску структуру и поновну употребу кода у систему.

* + 1. Node2D и наслеђивање

Већина чворова у 2D игри наслеђује чвор ***Node2D***, који обезбеђује основне функције као што су позиционирање, ротација и скалирање у 2D простору. Такође омогућава контролу редоследа приказивања при рендеровању.

Чворови који наслеђују ***Node2D*** обухватају различите типове елемената као што су ***Sprite2D***, који се користи за приказивање слика и спрајтова, и ***CharacterBody2D***, који омогућава контролу физике и колизије објеката у игри. Ова структура наслеђивања омогућава велику флексибилност у дизајну игара, јер сваки нови елемент може наследити постојеће функционалности и прилагодити их према потребама игре.

* + 1. Сигнали

**Сигнал** је механизам који омогућава објектима да комуницирају међусобно на ефикасан и разматљив начин. Сигнали раде по принципу емитовања и повезивања, омогућавајући да један објекат обавести друге објекте о неким догађајима без директне зависности или позивања њихових функција. Ово помаже да се смањи повезаност између објеката и омогућава већу флексибилност у дизајну.

* + 1. GDScript

***GDScript*** је програмски језик високог нивоа, објектно оријентисан, императиван и постепено типизиран, направљен за *Godot*. Користи синтаксу засновану на увлачењу, сличну језицима као што је *Python*. Његов циљ је да буде оптимизован за и тесно интегрисан са *Godot Engine*-ом, омогућавајући велику флексибилност при креирању садржаја и интеграцији.

Током развоја ове игре, ***GDScript*** је коришћен за програмирање логике кретања играча, управљање непријатељима, интеракције са окружењем, као и за имплементацију корисничког интерфејса и других елемената игре. Због своје брзине и флексибилности, GDScript је омогућио брзе итерације и тестирање нових функција.

Скрипте, тј. фајлови у којима је написан код, могу постојати самостално или се могу доделити чворовима или сценама и тако управљати њиховим ресурсима. Такође у скрипти је могуће навести да је нека променљива **експортована** помоћу кључне речи @export што омогућава њено директно подешавање из самог едитора.

* 1. Развојно окружење

Током развоја игре, коришћен је ***Godot*-ов уграђени едитор** за већину послова, укључујући креирање сцена, управљање чворовима и подешавање пројекта. За писање скрипти, коришћено је интегрисано развојно окружење (IDE) **Visual Studio Code**, које је омогућило лако писање *GDScript* кода. Поред тога, коришћени су и плугини за синтаксну проверу како би се осигурало да је код чист и без грешака.

* 1. Контрола верзија

За управљање верзијама на пројекту коришћен је **Git** заједно са **GitHub** платформом, уз коришћење **GitHub Desktop** алата. Овај систем за контролу верзија омогућио је да се сви кораци у развоју прате и да се врати на претходне верзије у случају потребе. **GitHub Desktop** је поједноставио процес контроле верзија, а **GitHub** платформа је омогућила чување резервних копија пројекта.

1. Општа подешавања

Током развоја, коришћена је *autoload* скрипта названа ***GlobalScript***, која служи за складиштење глобалних података и управљање функционалностима које се користе у различитим деловима игре. Ово омогућава да важни елементи буду доступни свим сценама без потребе за експлицитним прослеђивањем референци. *Autoload* скрипте се понашају као *singleton* објекти.

**Колизијске маске и слојеви** су важан аспект управљања интеракцијама између објеката у игри.

* **Колизијски слојеви** дефинишу на којим слојевима се објекат налази, односно где тај објекат може бити детектован од других објеката.
* **Колизијске маске** одређују са којим другим слојевима објекат може ступити у интеракцију. На пример, објекат може бити на слоју "ENEMIES", али његова колизијска маска може бити подешена тако да реагује само на "PLAYER" и "WALLS", игноришући друге непријатеље.

У овом пројекту, колизијске маске су подешене на следећи начин:

* WALLS – слој 1, за све елементе који представљају препреке у игри.
* **PLAYER** – слој 2, за играча.
* ENEMIES – слој 3, за непријатеље.
* HEALTHBOX – слој 4, за објекте који се односе на здравље.

Ова подела омогућава јасну организацију интеракција између различитих елемената игре.

Улазни подаци (inputs) су такође конфигурисани тако да омогуће интуитивно управљање:

* Кретање је подешено на **WASD** тастере.
* Пуцање је додељено тастеру **K**.

Сва подешавања су статички типизирана, што значи да се типови података морају дефинисати унапред. Ово не само да омогућава бољу контролу и прецизну проверу грешака током развоја, већ и убрзава извршавање кода, јер компајлер зна који типови података се користе и може их оптимизовати.

Подешавања дисплеја су следећа:

* Резолуција: **1920x1080** пиксела.
* Режим: **Windowed**.
* Stretch mode: **Canvas\_items**.
* Stretch aspect: **Expand**.

1. Имплементација *HealthBox* и *DamageBox* система

Циљ је био направити модуларни спрегнути ситем који симулира прављење штете и смањивање животних поена ентитета. У оваквој имплементацији се лако могу додати функционалности одређеним ентитетима тако што се у хијерархију њихове сцене само дода ***HealthBox***, ако је циљ да ентитет има животне поене, односно дода се ***DamageBox***, ако је циљ да ентитет или област ствара штету.

* 1. HealthBox

Ова сцена наслеђује чвор ***Area2D***, што значи да представља регион 2D простора који је дефинисан децом чворовима који су типа ***CollisionShape2D***, такође има могућност да детектује када други објекти типа ***CollisionShape2D*** улазе или излазе из тог региона.

Садржи две променљиве, max\_health и curr\_health, прва представља максималне, а друга тренутне животне поене. Прва променљива је **експортована** да би се олакшало дизајнирање ентитета који садрже ову сцену. Има дефинисан **сигнал** који се зове health\_deplted. Поседује функцију за смањивање животних поена и када ти животни поени дођу до нуле или испод нуле емитује се наведени сигнал.

* 1. DamageBox

Ова сцена такође наслеђује чвор ***Area2D***. Садржи две експортоване променљиве, прву, should\_hurt\_player, која назначава да ли се штета прави играчу или непријатељима, и другу, damage, која садржи вредност штете у виду реалног броја.

У функцији \_ready(), коју садржи сваки чвор и која се позива када се чвор и његова деца учитају у стабло сцена, на основу променљиве should\_hurt\_player подешавају се вредности колизионих маска саме сцене ***DamageBox***.

***Area2D*** садржи већ уграђене **сигнале** за детекцију других ***Area2D*** и ***PhysicsBody2D***, па ћемо лако повезати повратне функције \_on\_area\_entered и \_on\_body\_entered са та два сигнала помоћу едитора. Прва функција служиће за детекцију објеката типа ***HealthBox***, а друга за детекију самог играча, јер играч заправо наслеђује ***CharacterBody2D***. У обе функције позива се одговарајућа функција самог објекта који је детектован за умањивање животних поена. У обе се такође емитује **сигнал** по имену damage\_collided који касније могу користити родитељи ове сцене.

1. Имплементација пројектила

У игри, пројектили су кључни елементи који омогућавају играчу и непријатељима да наносе штету другим објектима. Систем пројектила је дизајниран тако да буде флексибилан и лако проширив, омогућавајући различите типове пројектила са специфичним брзинама, путањама и количинама штете. За омогућавање проширивости изабран је полиморфизам објектно орјентисане парадигме.

* 1. ProjectileBase

Сцена ***ProjectileBase*** наслеђује чвор ***Node2D***. Као децу садржи ***DamageBox***, који садржи ***CollisionShape2D***, и садржи ***Sprite2D***. У скрипти има дефинисане променљиве speed, која је типа float, direction, која је типа Vector2 и референцу на свој ***DamageBox*** подназивом damage\_box. Референца се дохвата помоћу кључне речи ***@ready*** испред имена променљиве, која указује да ће променљива бити иницијализована тек након што је чвор учитан у сцену, и оператора ***$*** након којег је наведено име чвора који припада сцени.

Функција \_physics\_proces прима параметар delta, који предствља време протекло од прошлог извршавања фунцкије и наслеђена је из Node2D. Ова функција се извршава у тачно дефинисаним интервалима од шездесет пута у секунди и користи се за физичке симулације. У овој сцени наведена функција је надјачана. У њој се само позива функција move\_projectile којој прослеђује параметар delta и коју наследници ове сцене треба да надјачају. Функција move\_projectile рачуна кретање пројектила на једноставан начин тако што на позицију додаје производ брзине, нормализованог вектора кретања и временског интервала. То се ради у случају да пројектил нема превише специфичан начин кретања, у том случају довољно је подесити му вектор и брзину.

У иницијализацији сцене, дакле у функцији \_ready, функција \_on\_damage\_collided се повезује на **сигнал** damage\_collided којег емитује damage\_box. У тој функцији позива се queue\_free која уклања сцену из стабла сцена, односно уклања објекат из игре.

При дизајнирању пројектила потребно је урадити следеће:

* Наследити сцену ***ProjectileBase*** и уклонити скрипту која је закачена, то се ради јер ће бити закачена скрипта родитељске сцене.
* Потребно је закачити нову скрипту и написати да она наслеђује ***ProjectileBase***
* Изабрати колозиони облик и подесити му величину.
* Убацити одговарајући визуелни спрајт.
* У општем случају подесити вектор и брзину кретања
* У зависности од тога да ли пројектил припада играчу или непријатљу потребно је подесити damage\_box на начин који је раније описан.
* По потреби додати специфичну логику у скрипту.

У наставку се могу видети неколико примера пројектила.

* 1. ProjectileEnemySine

Поред уобичајеног кретања помоћу вектора кретања и брзине, овај пројектил осцилира, односно има додатну компоненту у кретању која је нормална на његов вектор кретања, и та компонента брзине осцилира синусоидално.

* 1. ProjectileEnemyTarget

Овај пројектил дохвата позицију играча и поставља свој вектор кретања ка њему.

1. Имплементација пуцања код непријатеља

Пуцање је генерализовано помоћу сцене која се зове ***Shooter***. Ова сцена наслеђује чвор ***Node2D***. У себи садржи чвор који је типа ***Timer***. Кључна променљива је projectile\_scene, којa је типа ***PackedScene***, тај тип представља апстракцију серијализоване сцене и може му се доделити сцена као вредност. Садржи функцију shoot() која је одговорна за логику пуцања.Од променљивих такође садржи shoot\_cooldown, која представља интервал између два пуцања и референцу на ***Timer*** под називом shoot\_timer који емитује **сигнал** на који је потписана функција \_on\_timer\_timeout().

Када чвор уђе у сцену, у функцији \_ready се подешава време чекања тајмера и он се стартује. Након истека тајмера, сигнал \_on\_timer\_timeout покреће функцију \_on\_timer\_timeout која позива shoot, која инстанцира нови пројектил типа ***ProjectileBase*** и поставља га на позицију пуцача. Након тога, тајмер се поново покреће, чиме се омогућава континуирано пуцање у интервалима.

1. Имплементација непријатеља