VISOKA ŠKOLA STRUKOVNIH STUDIJA

ZA INFORMACIONE I KOMUNIKACIONE TEHNOLOGIJE

INTERNET TEHNOLOGIJE

**Razvoj “game engine-a” u Angular-u**

Završni rad

Mentor: Student:

Dr. Nenad Kojić, dipl inž Uroš Ćirić 50/15

Beograd, 2019.

VISOKA ŠKOLA STRUKOVNIH STUDIJA

ZA INFORMACIONE I KOMUNIKACIONE TEHNOLOGIJE

INTERNET TEHNOLOGIJE

Predmet: Web programiranje

Tema: **Razvoj “game engine-a” u Angular-u**

Ocena \_\_\_ ( \_\_\_\_\_\_\_\_\_ )

Članovi komisije:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Sadržaj

[1 Sadržaj 3](#_Toc8825010)

[2 Uvod 5](#_Toc8825011)

[3 Radno okruženje 6](#_Toc8825012)

[4 Organizacija 7](#_Toc8825013)

[4.1 HTML i CSS 8](#_Toc8825014)

[4.2 Engine 9](#_Toc8825015)

[4.3 Screen 11](#_Toc8825016)

[4.7 Entities 12](#_Toc8825017)

[4.10 Collision 12](#_Toc8825018)

[4.11 Controls 13](#_Toc8825019)

[4.12 Sound 13](#_Toc8825020)

[4.13 UI 13](#_Toc8825021)

[4.14 Entity 14](#_Toc8825022)

[5 Kodovi 16](#_Toc8825023)

[5.1 HTML i CSS 16](#_Toc8825024)

[5.1.1 app.component.html 16](#_Toc8825025)

[5.1.2 game-screen.component.html 17](#_Toc8825026)

[5.2 CSS 19](#_Toc8825027)

[5.2.1 Styles.css 19](#_Toc8825028)

[5.2.2 App.component.css 20](#_Toc8825029)

[5.2.3 Game-screen.component.css 22](#_Toc8825030)

[5.3 Engine 28](#_Toc8825031)

[5.4 Screen 34](#_Toc8825032)

[5.5 Entities 39](#_Toc8825033)

[5.6 Collision 43](#_Toc8825034)

[5.7 Controls 45](#_Toc8825035)

[5.8 Sound 49](#_Toc8825036)

[5.9 UI 50](#_Toc8825037)

[5.10 Entity 55](#_Toc8825038)

[5.10.1 Avatar 59](#_Toc8825039)

[5.10.2 Unit 60](#_Toc8825040)

[5.10.3 Player 61](#_Toc8825041)

[5.10.4 Enemy 63](#_Toc8825042)

[5.10.5 Bullet 65](#_Toc8825043)

[5.10.6 Pickup 66](#_Toc8825044)

[5.10.7 EnemyCollection 68](#_Toc8825045)

[5.10.8 AmmoCollection 68](#_Toc8825046)

[5.10.9 PickupCollection 68](#_Toc8825047)

# Uvod

Kada se spomene razvoj igara, programski jezici koji padaju na pamet su C++, C#, Java, Python i njihove varijacije, uglavnom korišćeni u određenom razvojnom okruženju za pravljenje igrica („game engine“).

Game engine u sebi sadrži suštinsku logiku funkcionalnosti kao što su prikaz 2D i 3D grafike, fiziku, detekciju sudaranja, zvuk, animaciju, veštačku inteligenciju... Neki od popularnijih engine-a kao što su Unreal i Unity zadovoljavaju potrebu razvijanja većine igara. Danas su retki slučajevi gde pravljenje neke igre zahteva pisanje sopstvenog engine-a, takvi slučajevi bi zahtevali veći razvojni tim.

Ovaj projekat sam odlučio da pišem u TypeScript-u - proširenju JavaScript-a, a za okruženje sam odabrao Angular. Za prikaz 2D grafike će biti zaslužan HTM5 Canvas. Od malobrojnih engin-a za browser igre odlučio sam se da ne koristim nijedan, već u osnovi, da napišem svoj.

Ideja jeste korišćenje manje zastupljenih tehnologija i gotovih kodova kako bi se bolje razumela osnova i suština pri radu na sličnim projektima.

# Radno okruženje

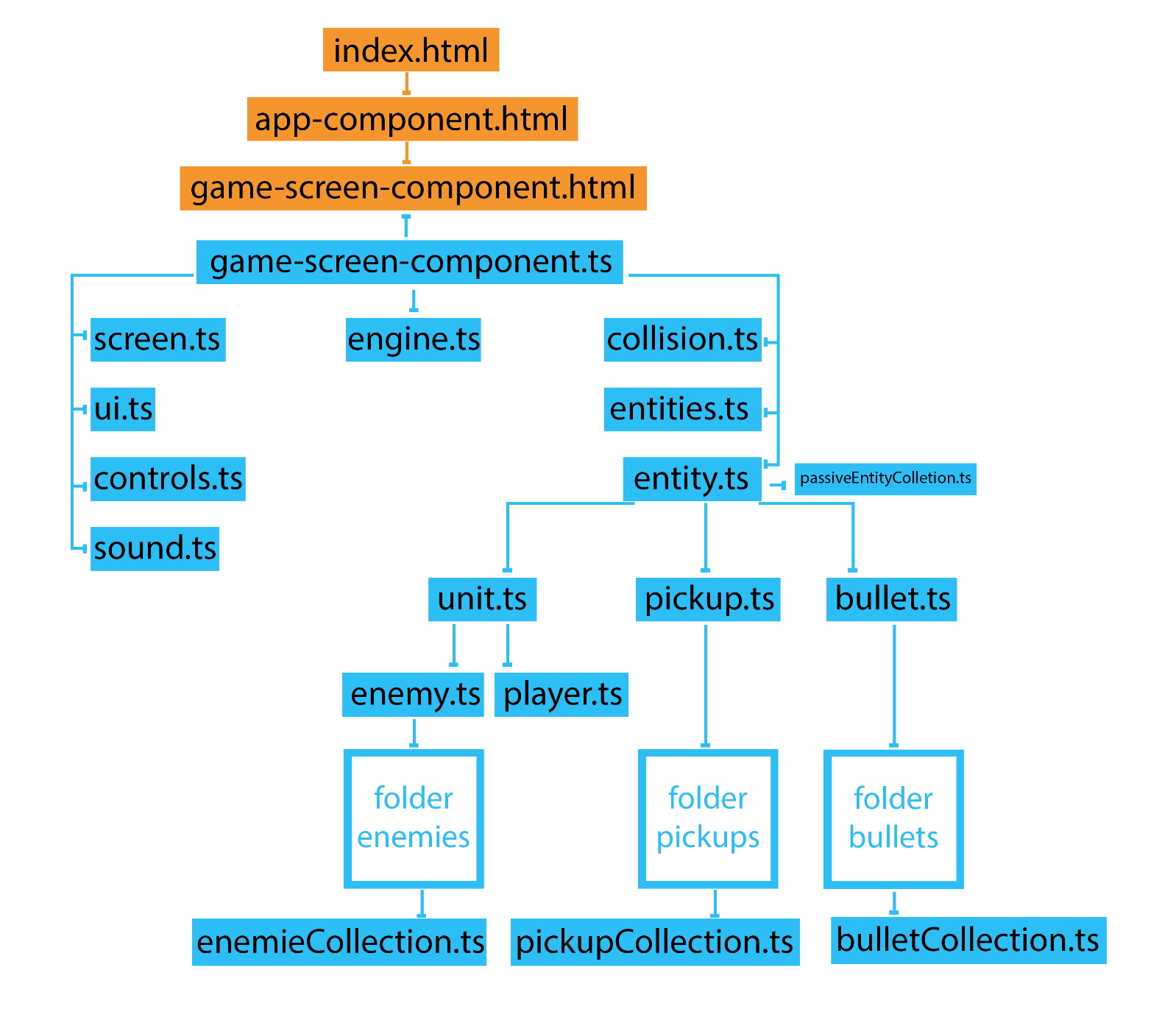
**Pri razvoju igrice korišćeno je razvojno okruženje:**

* Visual Studio Code

**Programski jezici koji su korišćeno za razvoj:**

* HTML5
* CSS3
* Sass
* TypeScript
* Angular 6.1.5

# Organizacija



U folderima enemies, pickups i bullets, nalaze se podklase tih roditelja (enemy1.ts, enemy2.ts, enemy3.ts…). Većina linija predstavlja uzajamnu parent-child vezu.

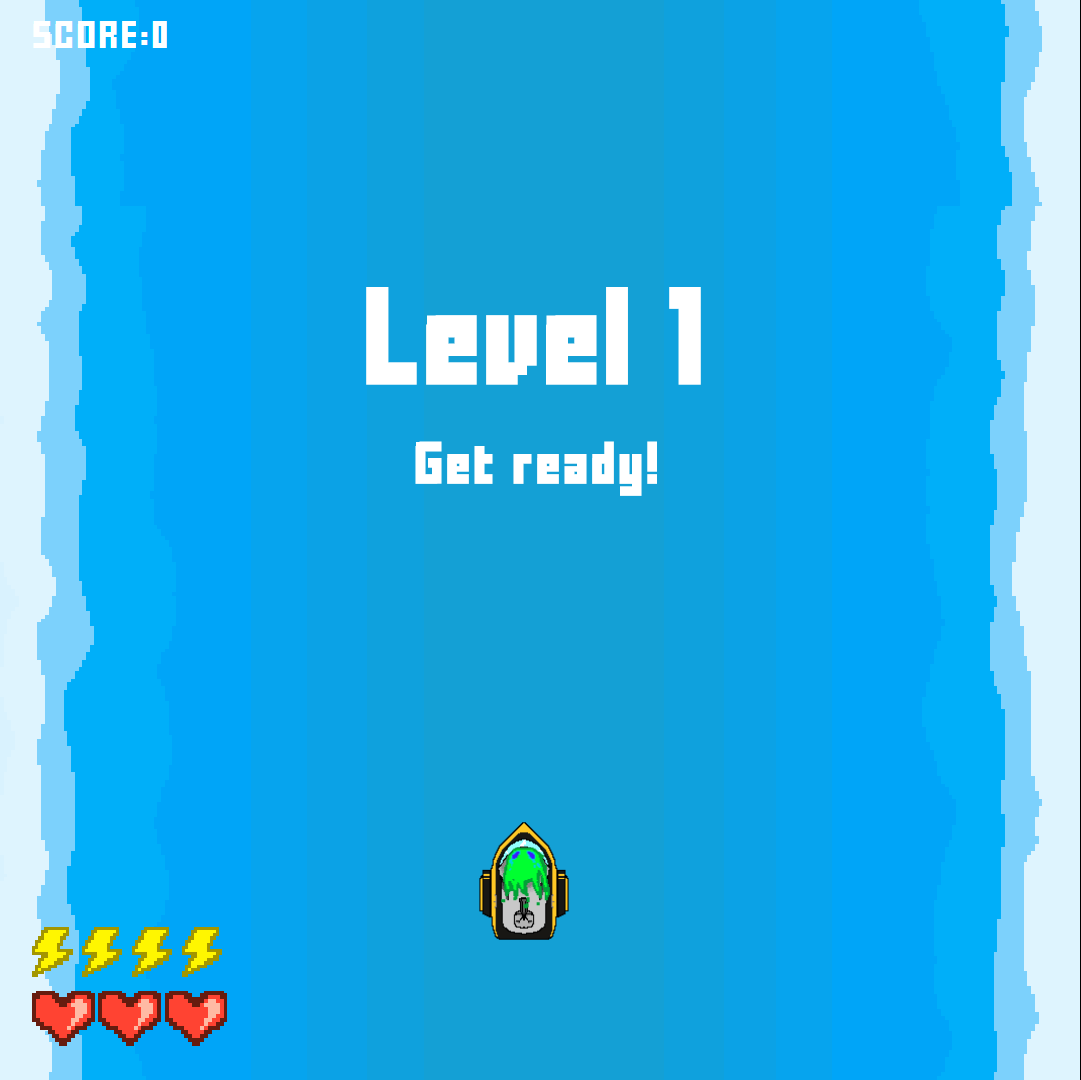
* 1. HTML i CSS

Kako se ovde html i css minimalno koriste, najlakše i najbrže je objasniti njih.

HTML se sastoji od par divova koji se ponašaju kao container za kanvas. Više kanvasa se nalaze jedan iznad drugog. Među ovim divovima, nalaze se još inicijalno sakriveni ekrani koji se pojavljuju u zavisnosti da li je igrica pauzirana ili završena. Pored njih još se nalaze i indikatori za rezultat igre, kao i životni i nergetski poeni.

CSS obezbeđuje da prikladne stvari budu prikazane ili sakrivene i ređa određene sloveje u njihovo odgovarajuće mesto. Takođe je zaslužan za učitavanje stranog fonta i pojedinih slika.

Postoji još HTML i CSS elemenata, ali oni se isključivo kreiraju i upravljaju putem skripti. Primeri toga bi bili kreiranje audio elemenata za puštanje zvukova i promena pozicionih vrednosti za pozadinsku sliku.



Početni ekran

* 1. Engine

Engine je zaslužan za ključne operacije kao što su pokretanje i pauziranje igre, učitavanje i promenu nivoa.

Treba napomenuti da je struktura nivoa zapisana u odvojenim JSON fajlovima. Ta struktura sadrži:

* Naziv nivoa
* Poruku igraču koja se ispisuje pri učitavanju nivoa
* Pozadinu nivoa (više o tome u nastavku)
* Muziku koja se pušta dok se odvija igra
* Talase protivnika sa informacijama o tome gde, kad i u kom redosledu da se prikažu

Ovde se taj JSON fajl obrađuje i pretvara u objekat. Poštuje redosled odvijanja događaja.

U ovom fajlu se takođe nalazi detekcija aktivnosti korisnika i trenutno vreme u milisekundama – u zavisnosti od potreba, nekad se koristi to vreme, a nekad brojanje pomenjenih frejmova.

Praćenje aktivnosti korisnika je neophodno uslednovih novih izmena internet browsera gde neophodna njegova interakcija sa sajtom kako bi mogli da mu se putem JavaScript-a puste video snimci i zvukovi.

Primer strukture nivoa:

{

"title": "Level 1",

"subtitle": "Get ready!",

"background": [

{"src" : "river.png", "speed": 20, "width":288, "height": 512},

{"src" : "", "speed": 3, "height": 100, "width": 300, "size":50, "margin":0, "place": "right", "delay": 0}

],

"music":"soundtrack\_pop\_is\_just\_a\_matter\_of\_preference.mp3",

"waves":

[

{

"pause" : 2000,

"requireClear" : true,

"enemies":

[

{"id":1,"pause":1000, "pos": ["A14", "B14", "J14"]},

{"id":1,"pause":0, "pos": ["A5", "B5", "F5"]},

{"id":1, "pos": ["A23","B23", "F23"]}

],

"pickups":

[ ]

},

{

"pause" : 2000,

"requireClear" : true,

"enemies":

[

{"id":1,"pause":1000, "pos": ["A14", "B14", "J14"]},

{"id":1, "pos": ["A10", "B10", "F10"]},

{"id":1, "pos": ["A6","B6", "C6"]},

{"id":1, "pos": ["A18", "B18", "F18"]},

{"id":1, "pos": ["A22","B22", "C22"]}

],

"pickups":

[{"id":2,"pos": "a14", "pause":0}]

},

{

"pause" : 2000,

"requireClear" : true,

"enemies":

[

{"id":2,"pause":1000, "pos": ["A14", "B14", "J14"]}

],

"pickups": [ ]

}

]

}

* 1. Screen

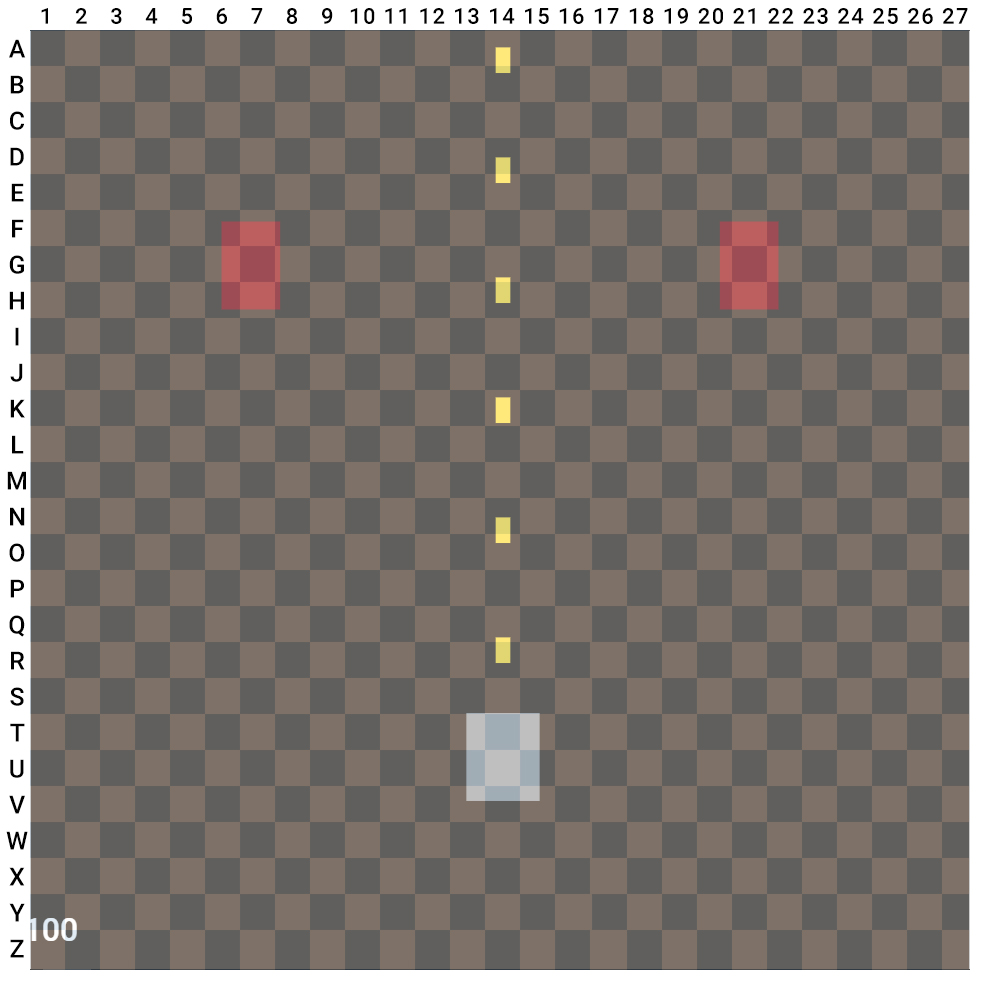
Ovde se nalazi sva logika zaslužna za vizuelni prikaz dešavanja. Ta logika podrazumeva sledeće:

## Upravljanje canvas-om

U ovom enginu postoje 3 canvas-a: po jedan za igrača, protivnike i municiju. Validan pristup bi bio i korišćenje jednog canvas-a, ali po maloj memorijskoj ceni, dobija se lakša kontrola. Pri učitavanju igre, oni se inicijalizuju. U slučaju promene dimenzije ekrana, canvas se prilagođava tako da su mu širina i dužina jednaki i da zauzima najviše mogućeg prostora.

Svaki canvas je zaslužan da renderuje – iscrtava svoje određene entitete. Kako većina monitora radi na 60 Hz, tako se i svaki canvas prazni i iscrtava 60 puta u sekundi.

## “Spawnovanje” – stvaranje protivnika i entiteta



Kako bi se omogućilo lakše stvaranje entiteta u koordinatnom sistemu, X i Y ose su podeljene na slova i brojeve, nalik obrnutoj šahovskoj tabli.

gridPos(**M14**) bi nam vratio poziciju blizu sredine, **B25** bi vratio gornji desni ugao, **X4** donji levi itd...

Vizuelna reprezentacija mreže za pozicioniranje entiteta

## Upravljanje pozadinskom slikom

Pozadina ovde se ne iscrtava u canvas-u već putem CSS-a i sastoji se iz više slojeva koji se kreću različitom brzinom kako bi dali efekat dubine prostora – “parallax”.

Pozadinski slojevi se kreću konstantnom brzinom. Najniži sloj mora da pokriva celu pozadinu i njegov početak i kraj moraju biti vertikalni odrazi kako bi slika mogla se nastavlja bez cepkanja.

* 1. Entities

Entities sadrži funkcije inicijalizacije i pomeranja entiteta koje se izvršavaju promenom svakog frejma kako bi se svaki entitet konstantno pomerao.

## Inicijalizacija protivnika i ostalih entiteta

Protivnicima se pri inicijalizaciji zadaju barem dve koordinatne tačke npr. B i C i one predstavljaju početnu i krajnju tačku putanje. Na te dve tačke izračunava se i treća tačka A koja koja je na istoj udaljenosti od tačke B koliko je i tačka B, samo orijentisana u suprotnom smeru. Time se postiže da se protivnik ne stvori na sred kanvasa, već da “uleti” sa najbliže tačke van njega.

## Pomeranje entiteta

U svakom frejmu entiteti se pomeraju. Protivnici se pomeraju po zadatoj liniji i stanu kada dođu na kraj, metkovi se uglavnom pomeraju vertikalno nadole ili nagore, stacionarni entiteti se pomeraju vertikalno nadole.

Radi čuvanja memorije briše se svaki entitet koji se pomeri van ekrana.

Pri ovom pomeranju protivnici takođe napadaju u određenom vremenskom intervalu.

* 1. Collision

Logika za detekciju sudaranja. Svaki entitet je predstavljen u formi pravougaonika. Ovde se ispituje da li u datom trenutku ivica jednog entiteta ne preklapa ivicu drugog.

U slučaju da se ivice entiteta preklapaju npr metka i protivnika, metak se briše (ako je tako podešen), a protivnik gubi toliko životnih poena koliko metak nosi poena za napad. U slučaju da protivnik nema preostale životne poene, briše se i on.

Pri brisanju entiteta, moguće je puštati zvukove i određene animacije tog entiteta.

* 1. Controls

Kod kontrola, sluša se aktivnost tastature i miša. Pri pomeraju miša, avatar igrača ubrzava ka kursoru. Ubrzavanje ovde je neophodno, da se avatar nalazi konstantno na kursoru, teže bi bilo sprečiti varanje u igri sa metodom gde se igra pauzira, zatim se pomeri miš i igra se nastavi. Takođe postoji izbor korišćenja strelica na tastaturi.

Levim klikom se aktivira specijalni potez.

U slučaju da se igrač pomeri van ekrana, avatar će ostati na samoj ivici.

* 1. Sound

Funkcija koja pravi HTML5 audio elemente.

Inicijalno su svi zvuci utišani dok korisnik ne interaktuje sa sajtom, u suprotnom zbog novih pravila internet pretraživača, zvuci ne bi mogli da se puštaju.

Zvuci se mogu puštati neprekidno kao pozadinska muzika ili jednom kao posledica stvaranja ili sudaranja entiteta.

* 1. UI

UI ili User Interface je zaslužan za sve ostalo što se prikazuje korisniku, a ne pripada kanvasu:

* Prikaz uvodnih video klipova
* Prikaz završnog ekrana u zavisnosti da li igrač pobedio ili izgubio
* Prikaz menija za vreme pauze
* Prikaz i promena životnih poena, energetskih poena i poena za finalni rezultat.
  1. Entity

Entity je glavna klasa za svaki entitet što se iscrtava na ekranu. Ti entiteti uključuju samog avatara koga igrač kontroliše, protivnike, prijateljske i neprijateljske napade, entitete koje igrač može da pokupi prelaskom preko njih i pasivne entitete koji uglavnom postoje iz estetskih razloga.

Podklase od entiteta su:

1. Unit
   1. Player
   2. Enemy
2. Bullet
   1. FriendlyBullet
   2. EnemyBullet
3. Pickups

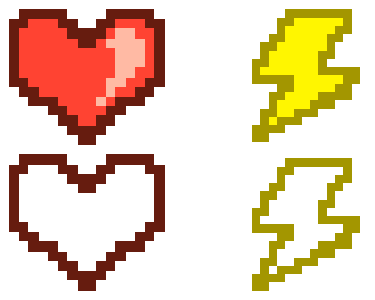
Konačno, ove podklase nasleđuju njihove podklase; primer: klasu Enemy nasleđuju klase Enemy1, Enemy2, Enemy3 itd....

Osobine koje sadrži svaki entitet su:

* Identifikacioni broj i naziv
* Stanje u kome je entitet (stvaranje, mirovanje, umiranje...)
* X i Y koordinatke gde se trenutno nalazi
* Visina i širina
* Brzina i ubrzanje
* Životni poeni i poeni za napad
* Niz zvukova koji se puštaju u prikladnom trenutku
* Podatke za animaciju
  + Spritesheet – niz frejmova za animaciju u jednoj slici
  + Visinu i širinu spritesheet-a

Svaki entitet u sebi ima funkcije za:

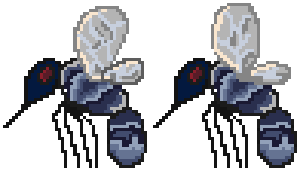
* Renderovanje ili crtanje
* Pomeranje
* Napadanje
* Animiranje
* Promene veličine u odnosu na ekran
* Puštanje zvuka



Puni prazno stanj životnih i energetskih poena



Primer sprite sheet animacije pucanja mehura



Animacija komarca u dva frejma

Protivnici i njihove varijacije

# Kodovi

* 1. HTML i CSS
     1. app.component.html

<div id="container">

<!-- Intro -->

<div id="videoContainer">

<p id="skipIntro">click anywhere to skip</p>

<video id="videoScreen" muted style="display:none;">

<source src="../../assets/Intro-1.mp4" type="video/mp4">

This browser is not supported.

</video>

</div>

<!-- The game -->

<app-game-screen id="theGame" style="display:none"></app-game-screen>

</div>

* + 1. game-screen.component.html

<div id="hud-container">

<div id="hud" classs="ui layer">

<div id="score-container">

<p>SCORE: </p><p id="scoreNumber">99999999</p>

</div>

<div id="stats">

<div id="energy-container">

<div id="fullenergy"></div>

</div>

<div id="heart-container">

<div id="fullheart"></div>

</div>

</div>

</div>

</div>

<div id="left-side" class="sidebars">

</div>

<div id="game-screen">

<canvas id="enemyScreen" class="layer"></canvas>

<canvas id="bulletScreen" class="layer"></canvas>

<canvas id="playerScreen" class="layer"></canvas>

<div class="screen-field-container">

<div id="screen-field">

<div id="pauseScreen" style="display:none;">

<div class="menuField" id="gamePaused">

<p>Game paused</p>

<br><br>

<p class="gameOptionsButton" id="settingsButton">Settings</p>

<p class="gameOptionsButton" id="creditsButton">Credits</p>

</div>

<div class="menuField" id="gameLost" style="display:none;">

<p>Aww man, you lost :(</p>

<br><br>

<a class="gameOptionsButton" href="javascript:window.location.href=window.location.href">Try Again</a>

<br>

<a class="gameOptionsButton" target="\_blank" href="https://giphy.com/embed/39manZbf7sISUlaRGG"

id="cryAgain">Cry again</a>

</div>

<div class="menuField" id="gameWon" style="display:none;">

<p>Wow, you won!</p>

<p>good job, thanks for playing</p>

<br><br>

<a class="gameOptionsButton" href="javascript:window.location.href=window.location.href">Play Again</a>

</div>

</div>

<div id="titlegroup">

<h1 id="leveltitle"></h1>

<h3 id="levelsubtitle"></h3>

</div>

</div>

</div>

<div id="backgrounds">

<div id="background-2" clas="game-background"></div>

<div id="background-1" clas="game-background"></div>

<div id="background-0" clas="game-background"></div>

</div>

</div>

<div id="right-side" class="sidebars">

</div>

* 1. CSS
     1. Styles.css

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

html,

body {

width: 100vw;

height: 100vh;

box-sizing: border-box;

}

\*,

\*:before,

\*:after {

box-sizing: inherit;

}

body {

font-family: 'Roboto';

background-repeat: no-repeat;

background-attachment: fixed;

}

.cleaner {

clear: both;

}

.clickable,

a {

cursor: pointer !important;

}

\*[disabled] {

cursor: not-allowed !important;

}

#container {

width: 100%;

height: 100%;

background-color: black;

position: relative;

}

* + 1. App.component.css

@font-face {

font-family: PixelFont;

src: url('../assets/fonts/FFFFORWA.TTF');

}

// Video

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

letter-spacing: 0.3vmin;

font-family: PixelFont;

}

html,

body {

width: 100%;

}

#videoContainer {

display: flex;

position: relative;

justify-content: center;

align-items: flex-start;

image-rendering: pixelated;

}

#videoScreen {

width: 100vmin;

height: 100vmin;

z-index: 999;

overflow: hidden;

object-fit: stretch;

image-rendering: pixelated;

}

@-webkit-keyframes pulse {

to {

color: rgba(255, 255, 255, 0);

}

}

@keyframes pulse {

to {

color: rgba(255, 255, 255, 0);

}

}

#skipIntro {

display: block;

color: rgba(255, 255, 255, 0.5);

font-family: PixelFont;

cursor: default;

position: absolute;

font-size: 3vmin;

bottom: 3vmin;

z-index: 9999;

-webkit-animation-name: pulse;

animation-name: pulse;

-webkit-animation-duration: 1.5s;

animation-duration: 1.5s;

-webkit-animation-timing-function: linear;

animation-timing-function: linear;

-webkit-animation-iteration-count: infinite;

animation-iteration-count: infinite;

-webkit-animation-direction: alternate;

animation-direction: alternate;

}

* + 1. Game-screen.component.css

\* {

-webkit-user-select: none;

/\* Safari \*/

-moz-user-select: none;

/\* Firefox \*/

-ms-user-select: none;

/\* IE10+/Edge \*/

user-select: none;

/\* Standard \*/

margin: 0;

padding: 0;

image-rendering: pixelated;

}

canvas {

display: block;

z-index: 2;

}

.layer {

position: absolute;

top: 0;

overflow: hidden;

}

.ui {

width: 100%;

height: 100%;

}

#hud-container {

width: 100%;

display: flex;

position: absolute;

justify-content: center;

z-index: 2;

#hud {

color: white;

height: 100vmin;

width: 100vmin;

font-size: 2rem;

#score-container {

font-family: PixelFont;

color: white;

padding-left: 3vmin;

p {

display: inline-block;

font-size: 2vmin;

}

}

// Stats

#stats {

padding: 3vmin;

position: absolute;

bottom: 0;

#heart-container {

width: 18vmin;

height: 5.25vmin;

background-image: url('../../assets/ui/heart.png');

background-repeat: repeat-x;

background-position-x: 0px;

background-position-y: -5.63vmin;

background-size: auto 200%;

#fullheart {

height: 100%;

width: 100%;

max-width: 100%;

min-width: 0%;

background-image: url('../../assets/ui/heart.png');

background-repeat: repeat-x;

background-position-x: 0px;

background-position-y: 0px;

background-size: auto 200%;

}

}

#energy-container {

width: 18vmin;

height: 5.2vmin;

background-image: url('../../assets/ui/energy.png');

background-repeat: repeat-x;

background-position-x: 0px;

background-position-y: -5.2vmin;

background-size: auto 200%;

margin-bottom: 1vmin;

#fullenergy {

height: 100%;

width: 100%;

max-width: 100%;

min-width: 0%;

background-image: url('../../assets/ui/energy.png');

background-repeat: repeat-x;

background-position-x: 0px;

background-position-y: 0px;

background-size: auto 200%;

}

}

#hp {

#statHp {

color: inherit;

background-color: inherit;

}

}

}

}

}

#left-side,

#right-side {

height: 100%;

background-color: black;

position: absolute;

top: 0;

z-index: 1;

}

#left-side {

left: 0;

}

#right-side {

right: 0;

}

//Pause Screen

#pauseScreen {

font-family: PixelFont;

font-size: 5vmin;

width: 100vmin;

height: 100vmin;

padding-top: 5vmin;

background-color: rgba(0, 0, 0, 0.75);

position: absolute;

color: white;

text-align: center;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

z-index: 99999;

a,

p {

color: white;

text-decoration: none;

}

}

.gameOptionsButton {

cursor: pointer;

margin: 2vmin;

}

@-webkit-keyframes gameOptionsButton {

to {

-webkit-transform: scale(1.1);

transform: scale(1.1);

}

}

@keyframes gameOptionsButton {

to {

-webkit-transform: scale(1.1);

transform: scale(1.1);

}

}

.gameOptionsButton {

display: block;

-webkit-transform: perspective(1px) translateZ(0);

transform: perspective(1px) translateZ(0);

box-shadow: 0 0 1px rgba(0, 0, 0, 0);

}

.gameOptionsButton:hover,

.gameOptionsButton:focus,

.gameOptionsButton:active {

-webkit-animation-name: gameOptionsButton;

animation-name: gameOptionsButton;

-webkit-animation-duration: 0.5s;

animation-duration: 0.5s;

-webkit-animation-timing-function: linear;

animation-timing-function: linear;

-webkit-animation-iteration-count: infinite;

animation-iteration-count: infinite;

-webkit-animation-direction: alternate;

animation-direction: alternate;

}

.screen-field-container {

width: 100%;

margin: 0 auto;

position: absolute;

z-index: 9999;

display: flex;

justify-content: center;

}

#screen-field {

width: 100vmin;

height: 100vmin;

max-height: 100vmin;

overflow: hidden;

}

#titlegroup {

font-family: PixelFont;

color: white;

text-align: center;

position: relative;

margin-top: -20vmin;

}

#leveltitle {

padding: 3vmin;

display: block;

font-size: 7vmin;

}

#levelsubtitle {

display: block;

font-size: 3vmin;

}

#backgrounds {

width: 100vmin;

height: 100vmin;

margin: 0 auto;

z-index: 0;

display: flex;

justify-content: center;

position: relative;

overflow: hidden;

}

#backgrounds>\* {

width: 100vmin;

height: 100%;

background-repeat: repeat-y;

background-position: 0px 0px;

position: absolute;

background-size: 100vmin;

overflow: hidden;

}

#background-0 {

z-index: 1;

}

#background-1 {

z-index: 2;

background-repeat: no-repeat !important;

background-size: 100% auto !important;

height: 100vmin;

}

#background-2 {

z-index: 3;

}

* 1. Engine

import { spawnEnemy, spawnPickup } from './entities';

import { enemies } from './entities/enemies/enemyCollection';

import { pickups } from './entities/pickups/pickupCollection';

import { displayTitle, titleDisplayed, resetTitle, playOutro, outroVideoExecuted, gameOutcome } from './ui';

import { bg, setBackground, enemyScreen } from './screen';

import { BgMusic } from './sound';

import \* as debug from './debug';

declare let $: any;

var clock = new Date();

export var time = clock.getTime();

export var level;

export function updateTime() {

let clock = new Date();

time = clock.getTime();

}

export function delayTime(timeMs) {

return (time + timeMs);

}

export var status = "paused";

export function togglePause(pauseButton) {

status = pauseButton;

}

// Music plays if user is active

export function checkGameStatus() {

if (status == 'paused') {

//The game is paused

}

}

document.addEventListener('mouseover', function () {

if (!userActive) { userActive = true; BgMusic.play(); }

})

//! Music plays if user is active

export var userActive = false;

export var numberOfLevels = 2;

//The class of a current level

export class Level {

number;

progress;

title;

background;

music;

waves;

subtitle;

bg0; bg1; bg2;

status;

constructor(lvl: any) {

this.number = lvl.number

this.progress = 0;

this.title = lvl.title;

this.subtitle = lvl.subtitle;

this.background = lvl.background;

this.music = '../../assets/sound/background/' + lvl.music;

//setting screen background

this.bg0 = lvl.background[0] ? lvl.background[0] : null;

this.bg1 = lvl.background[1] ? lvl.background[1] : null;

this.bg2 = lvl.background[2] ? lvl.background[2] : null;

setBackground([this.bg0, this.bg1, this.bg2]);

//!setting screen background

lvl.waves.forEach(wave => {

wave.cleared = false;

wave.requireClear = wave.requireClear != undefined ? true : false;

wave.pauseBeforeSet = false;

wave.pauseAfterSet = false;

if (wave.pause != undefined) {

wave.pauseSet = false;

wave.pauseClear = false;

} else {

wave.pauseSet = true;

wave.pauseClear = false;

}

wave.enemies.forEach(enemy => {

enemy.initiated = false;

enemy.pauseClear = enemy.pauseClear != undefined ? enemyScreen.pauseClear : 0;

});

wave.pickups.forEach(pickup => {

pickup.initiated = false;

if (pickup.pause == undefined) { pickup.pause = 0; }

});

});

this.waves = lvl.waves;

this.status = 'ready';

}

}

export var currentLevel;

//Used for loading new level (1 to initialize the game)

export function changeLevel(lvlnum) {

$.ajax({

url: `../src/app/game/levels/${lvlnum}.json`,

type: 'GET',

success: function (level) {

level.number = lvlnum;

currentLevel = new Level(level);

resetTitle();

let loadMusic = new Promise(function (resolve, reject) {

BgMusic.src(currentLevel.music);

let musicLoaded = true;

if (musicLoaded) { resolve(true); } else { reject(false); }

});

loadMusic.then(function () {

BgMusic.play();

});

}

});

}

//The timeline and how events play out

export function playLevel(lvl) {

if (lvl != undefined && lvl.status == "ready") {

if (titleDisplayed) {

let wave = lvl.waves[lvl.progress];

//--------Spawning enemies -------------

let enemiesToSpawn = wave.enemies.length;

if (enemiesToSpawn > 0) {

wave.enemies.forEach((enemy, i) => {

let clock = new Date();

let time = clock.getTime();

if (!enemy.initiated) {

//console.log(time);

if (i == 0) {

spawnEnemy(enemy.id, enemy.pos);

enemy.initiated = true;

enemy.pauseTime = time + enemy.pause;

}

else if (wave.enemies[i - 1].pauseClear) {

enemy.pauseClear = false;

enemy.initiated = true;

spawnEnemy(enemy.id, enemy.pos);

enemy.pauseTime = enemy.pause == undefined ? 0 : enemy.pause + time;

}

else if (time >= wave.enemies[i - 1].pauseTime) {

wave.enemies[i - 1].pauseClear = true;

}

}

else {

enemiesToSpawn--;

}

});

}

//!--------Spawning enemies -------------

//---------Spawning pickups -------------

let pickupsToSpawn = wave.pickups.length;

if (pickupsToSpawn > 0) {

wave.pickups.forEach((pickup, i) => {

if (!pickup.initiated) {

if (i == 0) {

pickup.initiated = true;

spawnPickup(pickup.id, pickup.pos);

pickup.pauseTime = time + pickup.puse;

}

else if (wave.pickups[i - 1].pauseClear) {

pickup.initiated = true;

pickup.pauseClear = true;

spawnPickup(pickup.id, pickup.pos);

if (pickup.pause == undefined) pickup.pauseTime = time + pickup.pause;

}

else {

if (time >= wave.pickups[i - 1].pauseTime) {

wave.pickups[i - 1].pauseClear = true;

}

}

}

else {

pickupsToSpawn--;

}

});

}

//!---------Spawning pickups -------------

//--------Checking if wave is cleared -------------

let everythingSpawned: boolean = (enemiesToSpawn <= 0 && pickupsToSpawn <= 0) ? true : false;

let enemiesKilledIfRequired: boolean = ((wave.requireClear && enemies.length <= 0) || !wave.requireClear) ? true : false;

if (!wave.pauseSet) {

if (everythingSpawned && enemiesKilledIfRequired) {

//--------Setting pause after a wave --------------

wave.pauseTime = time + wave.pause;

wave.pauseSet = true;

//!--------Setting pause after a wave -------------

};

}

else if (wave.pauseSet && !wave.pauseClear) {

if (time >= wave.pauseTime) {

wave.pauseClear = true;

wave.cleared = true;

}

}

else if (wave.cleared == true) {

if (currentLevel.progress < currentLevel.waves.length - 1) {

//Send in the next wave

currentLevel.progress++;

}

else {

//Send in the next level

if (currentLevel.number < numberOfLevels) {

changeLevel(currentLevel.number + 1);

}

else {

// Game finished, you win!

if (!outroVideoExecuted) { gameOutcome.won(); }

}

}

}

}

//!--------Checking if wave is cleared -------------

else {

//---------Displaying title -------------

displayTitle(lvl.title, lvl.subtitle);

//!--------Displaying title -------------

}

}

}

* 1. Screen

import { getEntities } from './entities';

export var sw;// Screen width

export var sh;// Screen height

export var sp;// Size point - according to screen width

export var cbX;//Checkerboard point in width

export var cbY;//Checkerboard point in height

//Player canvas

export var playerCanvas: any;

export var playerScreen;

//Bullet canvas

export var bulletCanvas: any;

export var bulletScreen;

//Enemy canvas

export var enemyCanvas: any;

export var enemyScreen;

//Sides

export var leftSide: any;

export var rightSide: any;

export function init(playerCanvasParam, bulletCanvasParam, enemyCanvasParam, leftSideParam, rightSideParam, bg0, bg1, bg2) {

playerCanvas = playerCanvasParam;

playerCanvas.imageSmoothingEnabled = false;

bulletCanvas = bulletCanvasParam;

bulletCanvas.imageSmoothingEnabled = false;

enemyCanvas = enemyCanvasParam;

enemyCanvas.imageSmoothingEnabled = false;

leftSide = leftSideParam;

rightSide = rightSideParam;

bg.layer0.obj = bg0;

bg.layer1.obj = bg1;

bg.layer2.obj = bg2;

resize();

}

//Resize event

window.addEventListener("resize", resize);

function resize() {

//Landscape

if (window.innerWidth > window.innerHeight) {

sh = window.innerHeight;

sw = sh;

sp = sh / 1280;

//Setting the sides

let sideSpacing = (window.innerWidth - sw) / 2 + 'px';

leftSide.style.width = sideSpacing;

rightSide.style.width = sideSpacing;

//Centering the screen

playerCanvas.style.left = sideSpacing;

enemyCanvas.style.left = sideSpacing;

bulletCanvas.style.left = sideSpacing;

}

//Portrait

else {

sw = window.innerWidth;

sh = sw;

sp = sh / 1280;

}

//Checkerboard points

cbX = sw / 54;

cbY = sw / 52;

//Player canvas

playerCanvas.width = sw;

playerCanvas.height = sh;

playerScreen = playerCanvas.getContext("2d");

//Bullet canvas

bulletCanvas.width = sw;

bulletCanvas.height = sh;

bulletScreen = bulletCanvas.getContext("2d");

//Enemy canvas

enemyCanvas.width = sw;

enemyCanvas.height = sh;

enemyScreen = enemyCanvas.getContext("2d");

//Update screen point for entities

getEntities().forEach(entity => {

entity.changeSp(sp);

});

}

//Function to clear canvas

export function clearCanvas(screen, canvas) {

screen.clearRect(0, 0, canvas.width, canvas.height);

}

//Function to clear all canvases

export function clearScreen() {

clearCanvas(playerScreen, playerCanvas);

clearCanvas(bulletScreen, bulletCanvas);

clearCanvas(enemyScreen, enemyCanvas);

}

//Gets coordinates from spawn grid (src/assets/SpawnGrid.jpg)

export function letterToNumber(letter) {

switch (letter) {

case 'a': return 1;

case 'b': return 2;

case 'c': return 3;

case 'd': return 4;

case 'e': return 5;

case 'f': return 6;

case 'g': return 7;

case 'h': return 8;

case 'i': return 9;

case 'j': return 10;

case 'k': return 11;

case 'l': return 12;

case 'm': return 13;

case 'n': return 14;

case 'o': return 15;

case 'p': return 16;

case 'q': return 17;

case 'r': return 18;

case 's': return 19;

case 't': return 20;

case 'u': return 21;

case 'v': return 22;

case 'w': return 23;

case 'x': return 24;

case 'y': return 25;

case 'z': return 26;

}

}

export function gridPos(place: any) {

//If it is a single place

if (typeof place == "string") {

let numberX = parseInt(place.toString().substr(1)), letter = (place.toString().substr(0, 1)).toLowerCase();

let numberY = letterToNumber(letter);

let x = numberX \* cbX \* 2 - cbX;

let y = numberY \* cbY \* 2 - cbY;

return [x, y];

}

//If it's multiple places

else if (typeof place == "object") {

let coordinates = [];

place.forEach(place => {

let numberX = parseInt(place.toString().substr(1)), letter = (place.toString().substr(0, 1)).toLowerCase();

let numberY = letterToNumber(letter);

let x = numberX \* cbX \* 2 - cbX;

let y = numberY \* cbY \* 2 - cbY;

coordinates.push([x, y]);

});

return coordinates;

}

}

// ------------ Background --------------------

export var bg = {

layer0: { obj: undefined, speed: 0, height: undefined, width: undefined, pos: 0, delay: 0, },

layer1: { obj: undefined, speed: 0, height: undefined, width: undefined, pos: 0, delay: 0, },

layer2: { obj: undefined, speed: 0, height: undefined, width: undefined, pos: 0, delay: 0, },

layer(num) {

switch (num) {

case 0: return bg.layer0;

case 1: return bg.layer1;

case 2: return bg.layer2;

}

}

}

export function setBackground(bgLayer) {

bgLayer.forEach((bcg, i) => {

if (bcg != null && bcg != '' && bcg != undefined) {

bg.layer(i).obj.style.backgroundImage = "url('../../assets/sprites/maps/" + bcg.src + "')";

bg.layer(i).speed = bcg.speed;

bg.layer(i).height = bcg.height;

bg.layer(i).width = bcg.width;

bg.layer(i).delay = bcg.delay

if (i > 0) {

bg.layer(i).obj.style.width = bcg.size + '%';

bcg.margin == undefined ? bcg.margin = 0 : bcg.margin;

switch (bcg.place) {

case 'left': bg.layer(i).obj.style.left = bcg.margin + '%'; break;

case 'right': bg.layer(i).obj.style.right = bcg.margin + '%'; break;

}

}

}

});

}

export function moveBackground(bgLayer) {

bgLayer.forEach((bcg, i) => {

bcg.obj.style.backgroundPosition = '0px ' + bcg.pos + 'px';

let speed = sp \* bcg.speed;

if (i == 0) {

let height = (bcg.height \* bg.layer0.obj.offsetHeight) / bcg.width;

if (bcg.pos >= height) { bcg.pos = speed - (height - bcg.pos); } else { bcg.pos += speed; }

}

else {

if (bcg.pos >= bg.layer0.obj.offsetHeight) { bcg.pos = - bcg.height - bcg.delay } else { bcg.pos += speed; }

}

});

}

* 1. Entities

import \* as screen from '../game/screen';

//Entities

import { enemyType, enemies } from './entities/enemies/enemyCollection';

import { pickups, pickupType } from './entities/pickups/pickupCollection';

import { bulletType, enemyAttacks, friendlyAttacks } from './entities/ammo/ammoCollection';

import { Player } from './entities/player';

import { Avatar } from './entities/avatar';

//!Entities

export var player;

export var avatar;

export function getEntities() {

return player != undefined ? [player].concat(enemies, pickups, friendlyAttacks, enemyAttacks) : [];

}

export function init() {

player = new Player();

avatar = new Avatar(screen.sw / 2, screen.sh / 2);

}

//--------------------- FUNCTIONS ---------------------------------------------------------------FUNCTIONS-----------------------------------/

//Spawning enemies

export function spawnEnemy(id, place) {

let pos = screen.gridPos(place);

let A = { x: pos[0][0], y: pos[0][1] }, C = { x: undefined, y: undefined }

//Spawning enemies as a single position for input

if (pos.length <= 1) {

C.x = (A.x < screen.sw / 2) ? (-A.x) : (screen.sw + A.x);

C.y = (A.y < screen.sh / 2) ? (-A.y) : (screen.sh + A.y);

}

//Spawning enemies with an array of position paths to follow at spawn

else {

let B = { x: pos[1][0], y: pos[1][1] }

C.x = A.x - (B.x - A.x);

C.y = A.y - (B.y - A.y);

}

pos.unshift([C.x, C.y]);

let enemy = new enemyType[id](pos[0][0], pos[0][1]);

enemy.spawnpath.progress.finish = place.length;

enemy.spawnpath.places = place;

enemy.spawnpath.progress.finish = place.length;

enemies.push(enemy);

}

//Spawning pickups

export function spawnPickup(id, place) {

let pos = screen.gridPos(place);

let pickup = new pickupType[id](pos[0], -pos[1]);

//pickup.balloon = new Balloon(pos[0], -pos[1]);

pickups.push(pickup);

}

//Move everything that moves with the same speed of background

//Drawing all entities

export function drawEntities() {

//Draw enemies

for (let i = 0; i < enemies.length; i++) {

enemies[i].draw();

enemies[i].animate();

enemies[i].worth -= 0.1;

if (enemies[i].worth < 0) { enemies[i].worth = 0; }

}

//Draw friendly bullets

player.shoot();

for (let i = 0; i < friendlyAttacks.length; i++) {

friendlyAttacks[i].draw(); // Drawing the bullets

friendlyAttacks[i].animate(); // Drawing the bullets

friendlyAttacks[i].y -= friendlyAttacks[i].speed; //Moving the bullets

if (friendlyAttacks[i].y < 0) { friendlyAttacks.splice(i, 1); } //Removing bullets when they get off screen

//Removing bullets that have a lifetime

else if (friendlyAttacks[i].timetodie != undefined) {

let clock = new Date();

let currenttime = clock.getTime();

if (currenttime >= friendlyAttacks[i].timetodie) {

friendlyAttacks.splice(i, 1);

}

}

}

//Draw enemy bullets

enemies.forEach(enemy => {

enemy.shoot();

});

for (let i = 0; i < enemyAttacks.length; i++) {

enemyAttacks[i].draw(); // Drawing the bullets

enemyAttacks[i].y += enemyAttacks[i].speed; //Moving the bullets

if (enemyAttacks[i].y > screen.sh) { enemyAttacks.splice(i, 1); } //Removing bullets when they get off screen

}

//Draw pickups

pickups.forEach(pickup => {

pickup.draw();

pickup.animate();

pickup.balloon.draw();

pickup.balloon.animate();

});

}

//Update entities on screen resize

export function updateEntities() {

if (player != undefined) {

player.h = player.height \* screen.sp;

player.w = player.width \* screen.sp;

}

if (enemies != undefined) {

for (let i = 0; i < enemies.length; i++) {

enemies[i].h = enemies[i].height \* screen.sp;

enemies[i].w = enemies[i].width \* screen.sp;

}

}

}

//Moving enemies

export function moveEnemies() {

enemies.forEach((enemy, i) => {

if (enemy.status == "spawning") {

enemy.followSpawnPath();

}

//Deleting enemies below screen

if (enemy.y - enemy.height / 2 > screen.sh) { enemy.die(i); }

});

}

//Moving stationary items

export function moveStationary() {

let speed = 7.5 \* screen.sp;

pickups.forEach((pickup, i) => {

pickup.y += speed;

pickup.balloon.y += speed;

//Removing item if it's off screen

if (pickup.y - pickup.h > screen.sh) { pickups.splice(i, 1); }

});

}

//!--------------------- FUNCTIONS ---------------------------------------------------------------FUNCTIONS-----------------------------------/

* 1. Collision

import { player } from './entities';

import { pickups } from './entities/pickups/pickupCollection';

import { enemies } from './entities/enemies/enemyCollection';

import { friendlyAttacks, enemyAttacks } from './entities/ammo/ammoCollection';

import { updateScore } from './ui';

export function collides(entity1, entity2) {

if (entity1 == null || entity1 == undefined || entity2 == null || entity2 == undefined) {

return;

}

if (((entity1.x + entity1.w / 2 > entity2.x - entity2.w / 2 && entity1.x + entity1.w / 2 < entity2.x + entity2.w / 2) || (entity1.x - entity1.w / 2 > entity2.x - entity2.w / 2 && entity1.x - entity1.w / 2 < entity2.x + entity2.w / 2)) && ((entity1.y + entity1.h / 2 > entity2.y - entity2.h / 2 && entity1.y + entity1.h / 2 < entity2.y + entity2.h / 2) || (entity1.y - entity1.h / 2 > entity2.y - entity2.h / 2 && entity1.y - entity1.h / 2 < entity2.y + entity2.h / 2))) {

return true;

}

let temoraryEntity = entity1;

entity1 = entity2;

entity2 = temoraryEntity;

if (((entity1.x + entity1.w / 2 > entity2.x - entity2.w / 2 && entity1.x + entity1.w / 2 < entity2.x + entity2.w / 2) || (entity1.x - entity1.w / 2 > entity2.x - entity2.w / 2 && entity1.x - entity1.w / 2 < entity2.x + entity2.w / 2)) && ((entity1.y + entity1.h / 2 > entity2.y - entity2.h / 2 && entity1.y + entity1.h / 2 < entity2.y + entity2.h / 2) || (entity1.y - entity1.h / 2 > entity2.y - entity2.h / 2 && entity1.y - entity1.h / 2 < entity2.y + entity2.h / 2))) {

return true;

}

return false;

}

export function collisionDetection() {

//For every enemy

for (let j = 0; j < enemies.length; j++) {

//If player hits enemy

if (collides(player, enemies[j])) {

player.hurt(enemies[j].dmg);

}

//For every friendly bullet

for (let i = 0; i < friendlyAttacks.length; i++) {

//If bullet hits enemy

if (collides(friendlyAttacks[i], enemies[j])) {

enemies[j].hurt(friendlyAttacks[i].dmg); // Enemy hurt

//If enemy hp is lower than 0

if (enemies[j].hp <= 0) {

updateScore(enemies[j].worth);

enemies[j].die(j);

}

friendlyAttacks[i].onhit();

if (friendlyAttacks[i].dieonhit) { friendlyAttacks[i].die(i); } //Bullet death

}

}

//For every enemy bullet

for (let i = 0; i < enemyAttacks.length; i++) {

//If bullet hits enemy

if (collides(enemyAttacks[i], player)) {

player.hurt(enemyAttacks[i].dmg); // Player hurt

enemyAttacks[i].die(i); //Bullet death

}

}

}

//For every pickup

for (let i = 0; i < pickups.length; i++) {

//If pickup hits player

if (collides(player, pickups[i])) {

pickups[i].activate(i);

}

}

}

* 1. Controls

import { playerCanvas, sw, sh } from './screen';

import { player } from '../game/entities';

import { togglePause, status } from './engine';

document.addEventListener('keydown', keyDownHandler, false); //Listener to keydown events

document.addEventListener('keyup', keyUpHandler, false); //Listener to keyup events

document.addEventListener("mousemove", mouseMoveHandler, false); // Listening to mouse movement

document.addEventListener("click", detectLeftButton, false); // Listening to mouse click

export var MovementType = 'keyboard';

export var pauseDisabled = false;

export function disablePause() {

pauseDisabled = true;

};

//Mouse movement

export var mouseX;

export var mouseY;

export function mouseMoveHandler(e) {

mouseX = e.clientX - playerCanvas.offsetLeft;

mouseY = e.clientY - playerCanvas.offsetTop;

MovementType = 'mouse';

}

//Keyboard movement

var rightPressed = false;

var leftPressed = false;

var upPressed = false;

var downPressed = false;

export var escapePressed = false;

var lastPressed = [];

function detectLeftButton(evt) {

evt = evt || window.event;

if ("buttons" in evt && status!="paused") {

player.shootspecial();

}

}

function keyDownHandler(event) {

if (event.keyCode == 39) {

rightPressed = true;

}

else if (event.keyCode == 37) {

leftPressed = true;

}

if (event.keyCode == 40) {

downPressed = true;

}

else if (event.keyCode == 38) {

upPressed = true;

}

if (event.keyCode == 27 && !pauseDisabled) {

if (escapePressed == true) {

togglePause('unpaused');

escapePressed = false;

}

else {

togglePause('paused');

escapePressed = true;

}

}

MovementType = 'keyboard';

}

function keyUpHandler(event) {

if (event.keyCode == 39) {

rightPressed = false;

}

else if (event.keyCode == 37) {

leftPressed = false;

}

if (event.keyCode == 40) {

downPressed = false;

}

else if (event.keyCode == 38) {

upPressed = false;

}

MovementType = 'keyboard';

}

function playerMove(accel) {

if (rightPressed) {

player.x += player.speed \* accel;

lastPressed[0] = 'right';

}

else if (leftPressed) {

player.x -= player.speed \* accel;

lastPressed[0] = 'left';

}

if (downPressed) {

player.y += player.speed \* accel;

lastPressed[1] = 'down';

}

else if (upPressed) {

player.y -= player.speed \* accel;

lastPressed[1] = 'up';

}

}

function slowDown(accel) {

if (lastPressed[0] == 'right') { player.x += player.speed \* accel; }

else if (lastPressed[0] == 'left') { player.x -= player.speed \* accel; }

if (lastPressed[1] == 'down') { player.y += player.speed \* accel; }

else if (lastPressed[1] == 'up') { player.y -= player.speed \* accel; }

}

export function changeMovement() {

if (MovementType == 'mouse') {

//Mouse movement

mouseMoveHandler;

let differenceX = mouseX - player.x;

let differenceY = mouseY - player.y;

//Border limitations

if (Math.abs(differenceX) >= player.speed) {

player.x += differenceX > 0 ? player.speed : -player.speed;

}

else {

player.x = mouseX;

}

if (Math.abs(differenceY) >= player.speed) {

player.y += differenceY > 0 ? player.speed : -player.speed;

}

else {

player.y = mouseY;

}

}

else {

//Keyboard Movement

if (!rightPressed && !leftPressed && !downPressed && !upPressed && player.accel > 0) {

player.accel -= 20;

slowDown(player.accel / 100);

}

else if ((rightPressed || leftPressed || downPressed || upPressed) && player.accel < 100) {

player.accel = 100;

playerMove(player.accel / 100);

lastPressed = [];

}

else {

playerMove(player.accel / 100);

}

if (player.accel < 0) { player.accel = 0; }

if (player.accel > 100) { player.accel = 100; }

}

if (player.x < player.w / 2) { player.x = player.w / 2 }

else if (player.x + player.w / 2 > sw) { player.x = sw - player.w / 2 }

if (player.y < player.h / 2) { player.y = player.h / 2 }

else if (player.y + player.h / 2 > sh) { player.y = sh - player.h / 2 }

player.draw();

}

* 1. Sound

import { userActive } from './engine';

export function sound(src?) {

this.sound = document.createElement("audio");

src == undefined ? this.sound.src = '' : this.sound.src = src;

this.sound.setAttribute("preload", "auto");

this.sound.setAttribute("controls", "none");

this.sound.style.display = "none";

document.body.appendChild(this.sound);

this.src = function (src) {

this.sound.src = src;

}

this.play = function () {

if (userActive) { this.sound.play(); }

}

this.stop = function () {

this.sound.pause();

}

this.volume = function (volume){

this.sound.volume = volume;

}

}

//Background music

export var BgMusic = new sound();

BgMusic.volume(0.1);

* 1. UI

import { player } from './entities';

import \* as debug from './debug';

import { changeLevel, togglePause } from './engine';

import { skipVideoIntro, skipVideoOutro } from './debug';

import { HttpInterceptorHandler } from '@angular/common/http/src/interceptor';

import { disablePause } from './controls';

export var hpBar: any;

export var energyBar: any;

var gameScreen: any;

var pauseScreen: any;

export var videoSrc: any;

export var videoStage = 1;

declare let $: any;

export var scoreBoard, score = 0;

export function init(hpBarParam, gameScreenParam, pauseScreenParam, videoScreen, theGameElement, energyBarElement) {

hpBar = hpBarParam;

gameScreen = gameScreenParam;

pauseScreen = pauseScreenParam;

theGame = theGameElement;

energyBar = energyBarElement;

videoSrc = <HTMLElement>document.querySelector("#videoScreen > source");

gameWon = <HTMLElement>document.querySelector("#gameWon");

gameLost = <HTMLElement>document.querySelector("#gameLost");

gamePaused = <HTMLElement>document.querySelector("#gamePaused");

scoreBoard = <HTMLElement>document.querySelector("#scoreNumber");

updateScore(0);

initVideo(videoScreen);

cryAgain = <HTMLElement>document.querySelector("#cryAgain");

}

export function updateStats() {

hpBar.style.width = player.hp + '%';

energyBar.style.width = player.energy + '%';

player.energy += 0.025; // const rate of energy recharge;

}

//Hiding mouse cursor

export function updateUi(gameStatus) {

if (gameStatus == 'unpaused') {

gameScreen.style.cursor = 'none';

pauseScreen.style.display = 'none';

}

else if (gameStatus == 'paused') {

gameScreen.style.cursor = 'auto';

pauseScreen.style.display = 'inherit';

}

}

export var titleDisplayed;

export var titleDisplayProgress = -60;

let Title, SubTitle, TitleGroup;

export function displayTitle(title, subtitle) {

if (!debug.skipIntro) {

if (titleDisplayProgress == -60) {

TitleGroup = <HTMLElement>document.querySelector("#titlegroup");

Title = <HTMLElement>document.querySelector("#titlegroup > #leveltitle");

SubTitle = <HTMLElement>document.querySelector("#titlegroup > #levelsubtitle");

Title.innerHTML = title;

SubTitle.innerHTML = subtitle;

}

else if (titleDisplayProgress >= 380) {

titleDisplayed = true;

player.shootAllowed = true;

}

else {

TitleGroup.style.marginTop = titleDisplayProgress / 3 + 'vmin';

}

titleDisplayProgress++;

}

else {

titleDisplayed = true;

player.shootAllowed = true;

}

}

export function resetTitle() {

titleDisplayed = false;

titleDisplayProgress = -60;

}

// Intro video

export var video;

export var theGame;

function initVideo(videoElement) {

video = videoElement;

video.muted = true;

video.onended = function () {

//After the video ends

playNextVideo();

}

document.addEventListener("click", function () { skipClicked = true; playNextVideo() });

let playNextVideo = function playNextVideo() {

if (videoStage == 3) {

if (skipClicked) {

videoStage++;

video.muted = true;

video.style.display = "none";

theGame.style.display = "block";

togglePause("unpaused");

introEnded();

changeLevel(1);

}

}

else if (videoStage < 3) {

skipClicked = false;

videoStage++;

video.pause();

videoSrc.setAttribute('src', `../../assets/Intro-${videoStage}.mp4`);

if (videoStage == 3) { video.loop = true }

video.load();

video.play();

}

}

let skipClicked = false;

function introEnded() {

document.removeEventListener("click", function () { skipClicked = true; playNextVideo() });

}

//Play the video

if (skipVideoIntro) {

video.style.display = "none";

togglePause("unpaused");

changeLevel(1);

theGame.style.display = "block";

}

else {

video.style.display = "block";

video.play();

}

}

// Outro video

export var outroVideoExecuted = false;

export function playOutro(msg?) {

video.src = '../../assets/outro.mp4';

theGame.style.display = "none";

video.style.display = "block";

video.onended = function () {

//After the video ends

gameOutcome.won();

}

if (skipVideoOutro) {

//Debug skipping video outro

gameOutcome.won();

}

else {

video.play();

outroVideoExecuted = true;

}

}

var cryAgain = undefined;

var gameLost, gameWon, gamePaused;

export var gameOutcome = {

won() {

disablePause();

togglePause('paused');

gamePaused.style.display = "none";

gameWon.style.display = "block";

},

lost() {

disablePause();

togglePause('paused');

$.ajax({

url: `https://api.giphy.com/v1/gifs/random?api\_key=IH5OMHObs41e3oPajWu2hWD258XrAylr&tag=cry&rating=PG-13`,

type: 'GET',

error: function () { },

success: function (gif) {

cryAgain.href = gif.data.images.original.url;

console.log(cryAgain.href);

}

});

gamePaused.style.display = "none";

gameLost.style.display = "block";

}

}

export function updateScore(num) {

score += Math.round(num);

scoreBoard.innerHTML = score;

}

* 1. Entity

import { sp } from '../screen';

export class Entity {

id;

name; //Name of an entity

status;//Current status of an enemie - idle - dying - spawning

x; //X coordinate of where the entity is drawn

y; //Y coordinate of where the entity is drawn

w; //Entity hitbox width

width;

h; //Entity hitbox height

height;

speed; //Speed of an entity

sp;// Screen point for scaling

boxColor = "#AAAAAA"; //Hitbox color - used mostly for testing purposes

screen; //Canvas ("layer") where entity is drawn

accel = 100; //Acceleration of an entity 0-100

hp; //Health points

multiplier; // multiplier fo image scaling

sprite = new Image();

lifetime = undefined; // Life of an entity | number = miliseconds | 'endframe' = dies after animation finishes

timetodie = undefined;

sound = {

"spawn" : {

play(){

},

volume(volume:any){}

},

"death": {

play() {

// console.log(`Death sound missing`);

},

volume(volume:any){}

}

}

constructor() {

this.sp = sp;

}

setWidthHeightMultiplier(SpriteWidth, SpriteHeight, Multiplier) {

this.multiplier = Multiplier;

this.width = SpriteWidth \* Multiplier; this.w = SpriteWidth \* Multiplier \* this.sp;

this.height = SpriteHeight \* Multiplier; this.h = SpriteHeight \* Multiplier \* this.sp;

this.animation.w = SpriteWidth;

this.animation.h = SpriteHeight;

}

// Changes screen point on resize

changeSp(sp) {

this.sp = sp;

this.w = this.width \* this.sp;

this.h = this.height \* this.sp;

}

draw() {

this.screen.fillStyle = this.boxColor;

if (this.sprite.src != '') {

this.screen.drawImage(this.sprite, this.animation.x, this.animation.y, this.animation.w, this.animation.h, this.x - this.w / 2, this.y - this.h / 2, this.width \* this.sp, this.height \* this.sp);

}

else { this.screen.fillRect(this.x - this.w / 2, this.y - this.h / 2, this.width \* this.sp, this.height \* this.sp); }

}

hurt(dmg) {

this.hp -= dmg;

}

changestatus(status) {

this.status = status;

}

setlifetime(time) {

let clock = new Date();

let currenttime = clock.getTime();

this.lifetime = time;

this.timetodie = currenttime + this.lifetime;

}

changeOpacity(n) {

this.screen.globalAlpha = n;

}

animation = {

"row": 0,

"frame": 0,

"framesPerRow": 4,

"frameRowPos": 0,

"frameCount": 0,

"x": 0,

"y": 0,

"w": 32,

"h": 32,

"state": "idle",

"states": {

"idle": {}, "death": {}, "prefire": {}, "postfire": {}

}

}

animate() {

if (this.animation.frameCount > 60 / this.animation.states[this.animation.state].fps) {

let startFrameRowPos = this.animation.states[this.animation.state].startFrame - (this.animation.states[this.animation.state].startRow \* this.animation.framesPerRow - this.animation.framesPerRow) - 1;

this.animation.frame++;

if (this.animation.frameRowPos >= this.animation.framesPerRow - 1) {

if (this.animation.row >= this.animation.states[this.animation.state].endRow - 1) {

this.animation.row = this.animation.states[this.animation.state].startRow - 1;

}

else {

this.animation.row++;

}

if (this.animation.row > this.animation.states[this.animation.state].startRow - 1) {

this.animation.frameRowPos = 0;

}

else {

this.animation.frameRowPos = startFrameRowPos;

}

}

else {

this.animation.frameRowPos++;

}

if (this.animation.frame >= this.animation.states[this.animation.state].endFrame) {

this.animation.row = this.animation.states[this.animation.state].startRow - 1;

this.animation.frameRowPos = startFrameRowPos;

this.animation.frame = this.animation.states[this.animation.state].startFrame - 1;

}

this.animation.frameCount = 0;

}

else this.animation.frameCount++;

this.animation.x = this.animation.frameRowPos \* this.animation.w;

this.animation.y = this.animation.row \* this.animation.h;

}

state(state) {

this.animation.state = state;

this.animation.frame = this.animation.states[this.animation.state].startFrame - 1;

this.animation.row = this.animation.states[this.animation.state].startRow - 1;

this.animation.frameRowPos = this.animation.states[this.animation.state].startFrame - (this.animation.states[this.animation.state].startRow \* this.animation.framesPerRow - this.animation.framesPerRow) - 1;

}

}

* + 1. Avatar

import { Entity } from './entity';

import { playerScreen, sw, sh } from '../screen';

export class Avatar extends Entity {

constructor(x,y) {

super();

this.screen = playerScreen;

this.x = x==undefined? sw/2 : x;

this.y = y==undefined? sh/2 : y;

this.setWidthHeightMultiplier(55, 72, 1.2);

this.sprite.src = "../../assets/sprites/player/avatar.png";

this.animation.framesPerRow = 4;

this.animation.states.idle = { "startFrame": 1, "endFrame": 4, "startRow": 1, "endRow": 1, "fps": 7, }

}

move(x,y){

this.x = x;

this.y = y;

this.animate();

this.draw();

}

}

* + 1. Unit

import { Entity } from './entity';

import { bulletType } from './ammo/ammoCollection';

export class Unit extends Entity {

type;

ammo = undefined;

bulletInstance = undefined;

specialammo = undefined;

specialbulletInstance = undefined;

shootWait = undefined;

shootAllowed = false;

changeAmmo(ammoId) {

this.ammo = ammoId;

this.bulletInstance = new bulletType[ammoId];

}

changeSpecialAmmo(ammoId) {

this.specialammo = ammoId;

this.specialbulletInstance = new bulletType[ammoId];

};

hurt(dmg) {

this.hp -= dmg;

}

onshoot() {

}

}

* + 1. Player

import { Unit } from './unit';

import { playerScreen, sw, sh } from '../screen';

import { time, delayTime } from '../engine';

import { bulletType, friendlyAttacks } from './ammo/ammoCollection';

import { gameOutcome } from '../ui';

export class Player extends Unit {

inv; //invulnerability timer

energy = 100; // energy for shooting special atacks

constructor() {

super();

this.type = 'friend';

this.screen = playerScreen;

this.x = sw / 2;

this.y = sh - (this.h / 2 + 100 \* this.sp);

this.hp = 100;

this.speed = 20 \* this.sp;

this.draw();

this.changeAmmo(1);

this.changeSpecialAmmo(3);

this.shootAllowed = false;

this.setWidthHeightMultiplier(90, 118, 1.2);

this.sprite.src = "../../assets/sprites/player/boat.png";

this.animation.framesPerRow = 1;

this.animation.states.idle = { "startFrame": 1, "endFrame": 1, "startRow": 1, "endRow": 1, "fps": 1, }

}

hurt(dmg) {

let clock = new Date();

let time = clock.getTime();

if (this.inv == null) {

this.inv = clock.getTime() + 1000//Time in ms of invulnerability;

this.hp -= dmg;

//this.screen.filter = "invert(50%)";

}

else if (this.inv != null && time > this.inv) {

//this.screen.filter = "invert(100%)";

this.inv = null;

}

else {

//What to do if invulnerable

}

if (this.hp <= 0) {

this.die();

}

}

shoot() {

if (this.shootAllowed) {

if (this.shootWait == null || this.shootWait == undefined || time >= this.shootWait) {

if (this.ammo != null || this.ammo != undefined) {

let x = this.x;

let y = this.y - this.h / 2 - this.bulletInstance.h / 2;

let attack = new bulletType[this.ammo](x, y);

attack.type = "friendly";

friendlyAttacks.push(attack);

this.shootWait = delayTime(attack.interval);//Interval between shots

}

}

}

}

shootspecial() {

if (this.shootAllowed && this.energy >= this.specialbulletInstance.cost) {

if (this.specialbulletInstance.double) {

let x = this.x + this.w \* 0.7;

let y = this.y - this.h / 2 - this.bulletInstance.h / 2;

let attack = new bulletType[this.specialammo](x, y);

attack.type = "friendly";

friendlyAttacks.push(attack);

x = this.x - this.w \* 0.7;

attack = new bulletType[this.specialammo](x, y);

attack.type = "friendly";

friendlyAttacks.push(attack);

}

this.energy -= this.specialbulletInstance.cost;

}

}

heal(hp) {

this.hp = this.hp + hp >= 100 ? 100 : this.hp + hp;

}

die() {

this.sound.death.play();

gameOutcome.lost();

}

}

* + 1. Enemy

import { Unit } from '../unit';

import { enemyScreen, gridPos } from '../../screen';

import { enemyAttacks, bulletType } from '../ammo/ammoCollection';

import { time, delayTime } from '../../engine';

import { enemies } from './enemyCollection';

export class Enemy extends Unit {

dmg = 0;

speed = 0;

worth = 100; // How many scoreboard points the enemy is worth

spawnpath = {

"progress": {

"start": 0,

"finish": undefined

},

"places": undefined,

"speed": undefined,

"moveX": undefined,

"moveY": undefined

}

constructor(x?: number, y?: number) {

super();

this.type = 'enemy';

this.boxColor = '#a52929';

this.screen = enemyScreen;

this.x = x;

this.y = y;

this.status = "spawning";

this.spawnpath.speed = this.sp \* 110;

this.shootAllowed = true;

//this.sound.death = new sound("../../assets/sound/fx/wilhelmscream.wav");

}

die(i) {

enemies.splice(i, 1);

this.sound.death.play();

}

shoot() {

if (this.shootAllowed) {

if (this.shootWait == null || this.shootWait == undefined || time >= this.shootWait) {

if (this.ammo != null || this.ammo != undefined) {

let x = this.x;

let y = this.y + this.h / 2 + this.bulletInstance.h / 2;

let attack = new bulletType[this.ammo](x, y);

attack.type = 'enemy';

enemyAttacks.push(attack);

this.shootWait = delayTime(attack.interval);//Interval between shots

this.onshoot();

}

}

}

}

followSpawnPath() {

let B = { x: undefined, y: undefined, location: gridPos(this.spawnpath.places[this.spawnpath.progress.start]) };

B.x = B.location[0];

B.y = B.location[1];

let difference = { x: Math.abs(B.x - this.x), y: Math.abs(B.y - this.y) }

let moveSpeed = (this.sp / 5) \* this.speed;

if (this.spawnpath.moveX == undefined) { this.spawnpath.moveX = difference.x / this.spawnpath.speed }

if (this.spawnpath.moveY == undefined) { this.spawnpath.moveY = difference.y / this.spawnpath.speed }

if (difference.x <= this.speed && difference.y <= this.speed) {

this.spawnpath.progress.start++;

if (this.spawnpath.progress.start == this.spawnpath.progress.finish) { this.status = "idle"; }

}

else {

if (difference.x >= this.speed && this.x < B.x) { this.x += this.spawnpath.moveX \* moveSpeed }

else if (difference.x >= this.speed && this.x > B.x) { this.x -= this.spawnpath.moveY \* moveSpeed }

if (difference.y >= this.speed && this.y < B.y) { this.y += this.spawnpath.moveY \* moveSpeed }

else if (difference.y >= this.speed && this.y > B.y) { this.y -= this.spawnpath.moveY \* moveSpeed }

}

}

}

* + 1. Bullet

import { Entity } from '../entity';

import { bulletScreen } from '../../screen';

import { friendlyAttacks, enemyAttacks } from './ammoCollection';

export class Bullet extends Entity {

type;

double: boolean = false;

dieonhit = true;

screen = bulletScreen;

constructor() {

super();

this.boxColor = '#ffe14f';

}

interval;//interval of bullets generated per time unit

die(i) {

if (this.type == 'friendly') {

friendlyAttacks.splice(i, 1);

}

else if (this.type == 'enemy') {

enemyAttacks.splice(i, 1);

}

}

onhit(){

}

spawn(attack){

if (this.type == 'friendly') {

friendlyAttacks.push(attack);

}

else if (this.type == 'enemy') {

enemyAttacks.push(attack);

}

}

}

* + 1. Pickup

import { Entity } from '../entity';

import { playerScreen } from '../../screen';

import { pickups } from './pickupCollection';

import { player } from '../../entities';

import {sound} from '../../sound';

export class Pickup extends Entity {

balloon;

constructor() {

super();

this.screen = playerScreen;

this.sound.death = new sound(`../../assets/sound/fx/1up/MP3/1up 2 - Sound effects Pack 2.mp3`);

this.sound.death.volume(0.1);

console.log();

}

spawnBalloon() {

this.balloon = new Balloon(this.x, this.y, this.animation.h, this.animation.w, this.multiplier \* 1);

}

die(i) {

pickups.splice(i, 1);

}

activate(i) {

this.onPickup(player);

this.sound.death.play();

this.die(i);

}

onPickup(player) {

console.log(`This pickup does nothing ¯\\\_(ツ)\_/¯`);

}

}

export class Balloon extends Entity {

constructor(x, y, w, h, m) {

super();

this.x = x - w / 4; this.y = y - h / 4;

this.screen = playerScreen;

this.setWidthHeightMultiplier(50, 50, m);

this.sprite.src = "../../assets/sprites/pickups/bouble.png";

this.animation.framesPerRow = 1;

this.animation.states.idle = { "startFrame": 1, "endFrame": 6, "startRow": 1, "endRow": 1, "fps": 10, }

this.state('idle');

}

}

export class BalloonBurst extends Balloon {

constructor(x, y, w, h, m) {

super(x, y, w, h, m);

this.animation.framesPerRow = 6;

}

}

* + 1. EnemyCollection

import { ScooterGirl } from './ScooterGirl';

import { FatGuy } from './FatGuy';

import { Pirana } from './pirana';

// List of enemy types

export var enemyType = [

undefined,//0

ScooterGirl,//1

FatGuy,//2

Pirana,//3

];

export var enemies = [];//Array of existing enemies

* + 1. AmmoCollection

import { LightRound } from './LightRound';

import { Shotgun } from './shotgun';

import { Torpedo } from './torpedo';

import { Explosion } from './explosion';

export var bulletType = [

undefined,//0

LightRound,//1

Shotgun,//2

Torpedo, //3

Explosion,//4

];

export var enemyAttacks = [];//Array of enemy bullets and other attacks

export var friendlyAttacks = [];//Array of friendly bullets and special attacks

* + 1. PickupCollection

import { Heal1, Heal2, Heal3, Heal4, Heal5 } from './heal';

export var pickups = [];

export var pickupType = [

undefined,//0

Heal1,//1

Heal2,//2

Heal3,//3

Heal4,//4

Heal5,//5

]

# Zaključak

Za izbor ove teme zaslužni su ne samo njena jedinstvensot i neuobičajenost, već i izazov koji doprinosi. Odluka da sam pravim svoj engine i igricu podstakla me je da razmišljam o mnogobrojnim detaljima koji prate njihovo razvijanje.

Sa razlogom je veoma retko da se razvija neka opširnija igra u komercijalne svrhe bez pomoći saradnika. To je jedan dug proces koga je teško vremenski predvideti, pogotovo ako je uključeno razvijanje sopstvenog engin-a i izrada svojih asset-a.

# Literatura

1. <https://stackoverflow.com/>
2. [https://www.w3schools.com](https://www.w3schools.com/)
3. <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial>