Strafe Overdead - Documentatie tehnica

Contents

Tehnologii utilizate

Java

GLSL

Gradle

Maven

libGDX(+plugins)

Git

Tiled

Eclipse IDE

Visual Studio Code IDE

Android Studio IDE

NetBeans IDE

JProfiler

Launch4j

Install4j

Oxygen XML editor

Photoshop

Aseprite

Spine

Texture Packer

Skin Composer

Cerinte de sistem

Cerinte sistem

Cerinte minime Desktop:

4GB RAM

3GB ROM(repository)

1GB ROM(instalat)

Procesor 2.5GHz

Cerinte minime Mobile:

Android 11+

6GB RAM

A fost testat pe:

- Desktop(dual boot Windows+Linux distro Mint):

  Procesor AMD Ryzen 5600

  Placa Video Nvidia RTX 2070super

  32GB RAM
- Masina Virtuala(Linux distro Manjaro)
- Laptop(Linux distro Ubuntu)

  Procesor Intel celeron 1.8GHz

  4GB RAM
- Google pixel 6 (Android 13)
- Samsung Galaxy S21 (Android 12)

Structura Proiectului

Orice proiect **libGDX** poate fi generat cu setup tool-ul dedicat. Acesta genereaza dupa preferintele utilizatorului mai multe proiecte Java care au dependente intre ele in functie de platformele si plugin-urile selectate in setup. Pentru build s-a folosit **Gradle**, dar dependentele catre **LWJGL** sunt gestionate de libGDX cu Maven. In cazul Strafer, exista:

- proiectul master ce inglobeaza toate proiectele
- proiectul core avand pachetele comune pentru toate versiunile
- proiectele**launcher** pentru fiecare platforma Android si Desktop (entry point-ul aplicatiei)

Arhitectura

Launchers

Fiecare launcher determina rezolutia si setarile necesare sistemului pe care ruleaza, apoi instantiaza game class-ul Strafer din core

Strafer- Game class-ul in care se regaseste rendering loop-ul si unde sunt declarate constantele globale si obiecte statice:

- WORLD_WIDTH, WORLD_HEIGHT
- aspectRatio
- SCALE_FACTOR unitatea de masura principala(1 metru=32 pixeli)
- assetManager incarca si returneaza asseturi recursiv din folderul assets in timpul fazei de loading
- i18n traduce stringuri in functie de limba setata
- spriteBatch se ocupa cu rendering-ul sprite-urilor pe ecran
- tiledMapRenderer un OrthogonalTiledMapRenderer modificat pentru a evita probleme de texture bleeding, se ocupa de rendering pentru fisiere tmx, tiled maps
- inputManager decide ce tip de platforma si periferice sunt disponibile si seteaza ce tip de InputProcessor este necesar(Keyboard,Controller,Touchscreen). Input-ul nu este hardcodat, ci se incarca din fisierul preferences, adica fiecare tasta poate fi reconfigurata de user. libGDX permite de asemenea utilizarea acestor tipuri de input interschimbabil intre platforme(Controller, Keyboard pe mobile, touch pe desktop etc)
- worldCamera este camera care se plimba pe harta dupa entitati

- extendViewport este folosit de worldCamera pentru a determina portiunea vizibla din ecran, acesta se adapteaza pentru orice resize al windowului si orice aspect ratio
- uiManager un scene2d stage, contine toate meniurile si componentele de user interface
- uiCamera este o alta camera cu un screenViewport care separa ui-ul de gameworld
- loadingScreen
- settingsScreen
- titleScreen

gameScreen ecranul in care se petrece gameplay loop-ul. Aici este instantiat pe uiManager un HUD, adica user interface-ul prezent in timpul jocului(healthbar si input pentru mobile). Pe gameScreen se instantiaza un GameWorld, container-ul pentru toata lumea jocului.

- rayHandler se va ocupa de iluminarea obiectelor din physics engine
- box2dWorld este container-ul pentru physics engine (box2d) Pentru a crea Body-uri, Sensor-uri si Fixture-uri folosim clasa Box2dFactory
- tiledMap este harta propriu-zisa a jocului din care se incarca automat checkpoint-uri si pereti
- FIXED_TIME_STEP constanta folosita pentru a da update in physics engine la un interval constant de timp, lucru ce separa frame rate-ul de coliziuni si movement
- entityEngine este un PooledEngine din framework-ul Ashley. Acesta este un entity-component system ce permite management-ul entitatilor care sunt formate prin compunere de componente, asupra carora actioneaza diferite system-uri .Un system itereaza prin orice entity care contine compinentele specificate pe constructor si proceseaza datele acestuia. Astfel creste valoarea de reutilizare a compinentelor, evitand inheritance-ul. Acesta are un pattern de factory, adica poate returna prin metode Entity-uri, spre exemplu player-ul

Player-ul are mai multe componente:

- PlayerComponent folosit pentru controlarea playerului, acesta are un sensor fixture pe body folosit pentru a determina alte entity-uri din proximitate
- EntityTypeComponent indica ce tip de entity este playerul si in ce state se afla(idle,hit, walk etc)
- PositionComponent pozitia in lume
- MovementComponent directiile de deplasare active, viteza si forta de dash
- SpriteComponent imaginea entity-ului si dimensiunile acesteia
- AnimationComponent un Animation<Sprite> si un timer intern folosit pentru actualizarea animatiei
- Box2dComponent indica engine-ului ca entity-ul ia parte in coliziuni. Body-ul este creat dupa dimensiunile sprite-ului avand doua Fixture-uri: fingerprint-ul(partea solida a corpului) si hurtbox-ul(folosit pentru interactiunile de combat cu hitboxuri)
- HealthConponent entity-ul ia parte in combat si poate primi damage.
- CameraComponent camera poate focusa entity-ul

alte entity-uri pot avea:

- DetectorComponent detecteaza proximitatea cu playerul
- CheckpointComponent primeste un checkpoint action si este focusat automat de camera in zonele hartii care sunt points of interest
- AttackComponent are un hitbox si alte caracteristici ale unui atac precum damage tip de atac etc

System-uri

- AnimationSystem se foloseste de counterele din AnimationComponent si schimba SpriteComponent-ul cu frame-ul curent. Animatiile sunt incarcate din AnimationProvider pe baza EntityType-ului si al EntityState-ului. Animatiile sunt salave in spritesheet-uri si decodate cu un fisier atlas in care sunt salvate informatii despre toate frame-urile animatiilor. Astfel este necesar un singur png si un fisier atlas pentru toate animatiile.
- CheckpointSystem executa actiunile din checkpointuri cand sunt atinse
- RenderingSystem foloseste spriteBatch-ul si randeaza Sprite-urile la pozitiile entity-urilor dupa ce compara pozitia lor pe axa Y, astfel incat sprite-urile sa para ca pot fi unul in spatele celuilalt.
- HudSystem actualizeaza hud-ul cu informatii din player

- CameraSystem decide unde este focusata camera. Daca player-ul este in proximitatea unui grup de inamici camera focuseaza in mijlocul grupului. Daca exista un checkpoint in proximitate, acesta va fi focusat. Daca playerul nu are entity-uri cu CameraComponent in proximitate este focusat singur.
- HealthSystem verifica daca un hurtbox a primit damage si actualizeaza nivelul de health
- CombatSystem aplica efecte de knockback daca un entity a fost lovit

FilteredContactListener este un listener adaugat la box2dWorld care filtreaza coliziunile astfel incat sa determine cand s-a realizat un contact intre hitbox-uri/hurtbox-uri sau intre sensorul playerului si un detector