Jelle Principaal & Stein Bout – Design document

# Introduction

This document describes the implementation of lab 3. We decided on making a platformer with infinite world using configurable chunks. The goal for the player is to get as far as possible in the level to get the highest score which can be saved on a leaderboard. As the score increases so will the difficulty by spawning more difficult enemies which the player needs to avoid to keep playing.

# Design

## Data structures

### Game state

The game state holds all the information the game needs to perform. This is represented using a data type:

data GameState = MkGameState { gChunks :: ActChunks

, gPlayer :: Player

, gEnemies :: [AI]

, gIsPaused :: Bool

, gRandom :: Int

, gKeyPresses :: [Event]

, gEventData :: EventData

, gXOffset :: Float

, gBitMapData :: BitmapData

, gPossibleChunks :: [Chunk]

, gLoaded :: Bool

}

Chunks: this holds the currently loaded chunks which need to be rendered  
Player: The player  
Enemies: The current enemies that have spawned  
isPaused: Whether the game is paused  
Random: Contains a random number when needed  
XOffset: How far we are in the world  
BitMapData: Contains the tileset  
PossibleChunks: Contains the loaded chunks for usage in level generation  
Loaded: Whether all required assets have been loaded in already  
KeyPresses: Stores the keys pressed

### Player

The player is represented with another data type containing the coordinates of the player and the velocity of the player.

data Player = MkPlayer {

pos :: FloatCoord,

vx :: Float,

vy :: Float

vgrav :: Float,

hitbox :: Hitbox,

isDead :: Bool,

Sprite :: Rectangle

}

data Hitbox = MkHitbox {

start :: Coord,

end :: Coord

}

### Chunk

In order to create more interesting worlds which don’t just contain straight lines, these chunks are loaded from a file containing plain tekst, making it easy to create new chunks to explore.

data Chunk = MkChunk { chunkId :: Int,

length :: Int,

startPos :: Int,

endPos :: Int,

unloadPos :: Int,

platforms :: [Platform] }

### AI

The AI represents the enemies that the player needs to avoid. If enemies touch the player the game will end, an exception for this is when the player jumps on top of the enemy, this will cause the enemy to explode.

data AI aiType x y = MkAI AI\_type Float Float Hitbox

deriving (Eq)

data AI\_type = AI1 | AI2 --Possible more types if needed

deriving (Eq)

### General types

All other types that help out the types listed above, the names should be pretty self-explanatory.

type Seed = Float

type FloatCoord = (Float, Float)

type Coord = (Int, Int)

type Platform = (Coord, Coord)

## Computer movement

At the moment, we have planned for two types of AI. One works like a gooma: It goes in a direction until it hits an obstacle, then changes direction. The other will continuously chase the player.

## Interface

The interface is a GUI made via the Gloss library. The screen will change over time and react tot he keyboard inputs to move the player or jump. There’s text on the upper left of the screen displaying the current score.

## Implementation of the minimum requirements

**Player:** The player can control the player to move left, right, jump in order to progress the level

**Enemies:** There are two types of enemies that can spawn to add a risk to the player

**Randomness:** Both the spawning of enemies is random and the chunks that are loaded in are random

**Animation:** We plan on animating the player and enemies walking and to add an effect when destroying an enemy

**Pause:** By pressing P the isPaused variable toggles, when this variable is true, the game will freeze until the pause key is pressed again.

**Interaction with file system:** The game loads custom levels and saves highscores via files

## Implementation of the optional requirements

**Different enemies:** We have two types of enemies who behave differently  
**Custom levels:** It will be possible to add more chunks by adding more files to a level folder, levels can be easily made via a text editor