Pac-Man (Original Design document)

We will be recreating the game Pac-Man using haskell. In this document we will describe how we plan on writing the code and what architecture we will implement.

The data type GameState keeps all the information about the entire game. Everything from whether the game is paused or not to the map of the playing field is stored here. We will now discuss various information being stored inside the GameState.

data GameState = Gamestate {

menustate :: MenuState,

maze :: BottomLayer,

score :: Score,

level :: Int,

entities :: EntityRecord,

inputBuffer :: InputBuffer

settings :: Settings

}

**MenuState**

data MenuState = PauseMenu PauseMenuState | StartMenu StartMenuState | SettingMenu | Playing

The MenuState datatype stores if the game is currently in a menu, and if so, what kind of menu. There are a few options in the pause menu and the start menu. The PauseMenuState and StartMenuState represent what option is currently selected in that menu. This wil also be implemented in the SettingMenu later.

data PauseMenuState = ContinueOption | Settingoption | ExitToStartOption | ExitToDesktopOption

data StartMenuState = PlayOption | SettingOption | ExitOption

**Score**

type Score = Int

The Score datatype keeps track of the score

**BottomLayer**

type BottomLayer = Map Position BottomLayerContent

The datatype BottomLayer is a map in which a position can be entered. It will then return BottomLayerContent, which is a datatype to store what bottom layer content is present at that position. Bottom layer content is content of which only the position matters, and we do not need any other information about.

data BottomLayerContent = Wall | SmallDot | Bigdot | Empty

**Level**

level :: Int

The level keeps track of which level the player is on

**Entities**

entities :: EntityRecord

Entities in the game are stored in the datatype EntityRecord

data EntityRecord = MkEntityRecord {

player :: PlayerEntity,

enemies :: [EnemyEntity],

fruits :: [Fruit]

}

This stores the player, the enemies and the fruits in the level.

data PlayerEntity = MkPlayer {

playerPosition :: Position,

playerMovementDirection :: Direction,

powerState :: PowerState

}

The player stores a position, a movement direction and whether the player is in a power state or not.

data EnemyEntity = MkEnemy {

enemyPosition :: Position,

enemyMovementDirection :: Direction,

enemyMovementType :: EnemyMovementType,

enemyStatus :: EnemyStatus

}

An enemy basically stores the same things as a player, except it stores a status instead of a power state, which can be either alive, dead or scared. It also stores a movement type. The enemies in Pac-Man all move differently through the maze, so in this type it is stored which enemy they are, so they can move accordingly.

data Fruit = MkFruit {

fruitType :: FruitType,

fruitPosition :: Position

}

The Fruit datatype stores the type of fruit and the position of said fruit.

The datatype Position is used a lot in these datatype. The datatype Position will be implemented as follows

data Position = MkPosition {xposition:: Int, yposition::Int}

deriving (Eq, Ord)

We also make a class Positioned a to be able to get the position from all the data types that use it. We also create an agent to be able to set the position and get and set the direction. We make the typeclass Agent as well, because only certain entities need to be able to set a position or get or set a direction.

class Positioned a where

getPosition :: a -> Position

class (Positioned a) => Agent a where

setPosition :: a -> Position -> a

getDirection :: a -> Direction

setDirection :: a -> Direction -> a

This can then be implemented as follows

instance Positioned PlayerEntity where

getPosition player = playerPosition player

instance Agent PlayerEntity where

setPosition player newPosition = player {playerPosition = newPosition}

getDirection player = playerMovementDirection player

setDirection player newDirection = player {playerMovementDirection = newDirection}

instance Positioned Fruit where

getPosition fruit = fruitPosition fruit

**InputBuffer**

inputBuffer :: InputBuffer

The inputBuffer is also stored in the gamestate. This is where the input for the direction of the player is stored, so this can be executed as soon as it is possible. We store this in the GameState, because we want to be able to access this information fast and easy.

data InputBuffer = Neutral | Up | Down | Left | Right

**Settings**

For now we only store the volume of the game in the settings. It will be possible to add other things to the settings datatype if it is desired.

data Settings = Settings{

volume :: Int

}

**Requirements**

* Player

The player is stored in the EntityRecord. This is of type PlayerEntity

data PlayerEntity = MkPlayer {

playerPosition :: Position,

playerMovementDirection :: Direction,

powerState :: PowerState

}

The player uses the InputBuffer in the gamestate to change their direction as soon as possible.

* Enemies

Enemies are stored in the EntityRecord as well. This is stored as a list of EnemyEntity.

data EnemyEntity = MkEnemy {

enemyPosition :: Position,

enemyMovementDirection :: Direction,

enemyMovementType :: EnemyMovementType,

enemyStatus :: EnemyStatus

}

The enemies have a variable that the player does not have. This is enemyMovementType. This tells the game which enemy it is. Each enemy moves differently.

data EnemyMovementType = Blinky | Inky | Pinky | Clyde

* Randomness

The ghosts will choose a random direction to go to at each corner when they are frightened.

* Animation

Pac-Man will open and close his mouth while moving. The skirts of the ghosts will be waving as well. The moving entities move with one block at the time, and since we want smooth movement, all of that will be animated as well.

* Pause

Pausing will be possible because of the MenuState in the GameState. When a certain key is pressed, the MenuState will change from “Playing” to “PauseMenu PauseMenuState”, where PauseMenuState is there to select certain options in the menu.

* Interaction with the file system

We will make a way to save and play custom levels. For this we only need to be able to load the maze of type BottomLayer and the entities of type EntityRecord into the GameState. We also want to make a way to save your progress. We can do this by saving the entire GameState, and by providing a way to load this GameState back in again.

**Optional requirements**

We will make different levels and provide options to load custom levels. We will provide the option to load custom levels by loading the maze of type BottomLayer and the entities of type EntityRecord into the GameState. This is also how we plan to make the different levels for the game. We will do this using JSON. We also will save the full game state and load this again using JSON.

**Separating the pure and impure parts**

We will separate the pure and impure parts by making only the parts which need to be impure impure. We will then combine them using monad operators like bind and return, as well as the functor operator fmap.

**Abstraction**

We will use typeclasses to get the position of any entity in the game. We can use the class Positioned a to get the position of entities.

class Positioned a where

getPosition :: a -> Position

We can use the class Agent a to set the position and get and set the direction of said entities. This class is exclusively for moving entities. Pac-Man and the ghosts do move, but things like the dots and fruits do not, so the class Agent a is unnecessary for them.

class (Positioned a) => Agent a where

setPosition :: a -> Position -> a

getDirection :: a -> Direction

setDirection :: a -> Direction -> a