Method of Approach

Group 4

Jakob Mendelsohn, 337301

Gijsbert Havinga, 336115

Bernike de Olde, 329964

Marijn Metzlar, 331703

CSV1B

School of Communication, Media and IT

Serious Game Development (SGDE2)

Manno Bult

Groningen, 26.05.2015

Game Concept

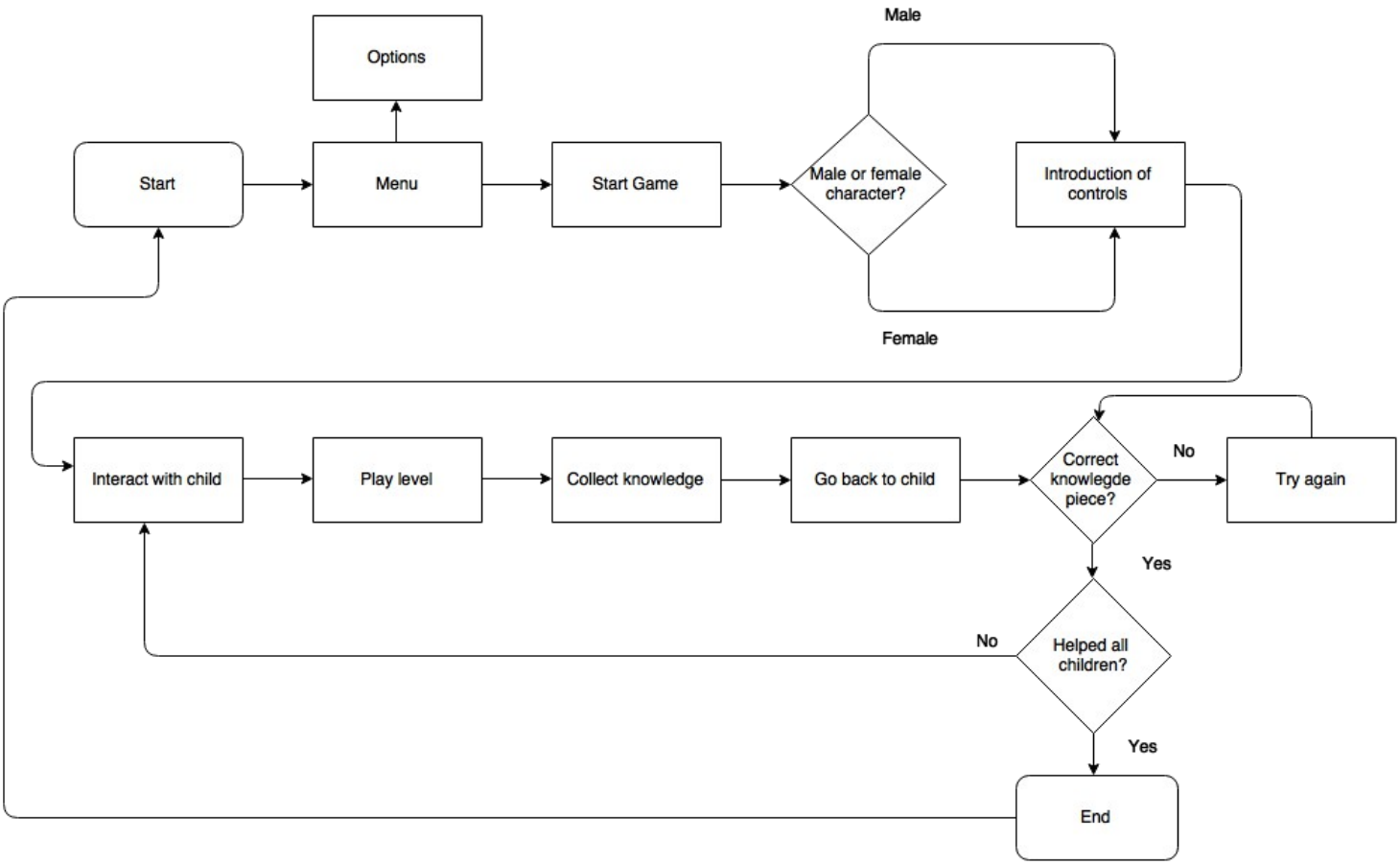
In our game the player has to collect knowledge in a number of individual levels and use this knowledge to help children with law-related problems. The game is divided into an overworld area where the player encounters several kids that have a problem and individual platforming levels, that the player has to complete in order to acquire the knowledge required to help a kid. If the player doesn't know what knowledge is needed to help a specific child he can ask his companion, which represents the KJRW, for help. The companion can also show the player the knowledge pieces he has collected including information on it. The levels vary in difficulty and it may sometimes be necessary to revisit a previous level in order to get previously unavailable or overlooked pieces of knowledge.

Technical Design

From this game concept a few technical requirements come to mind immediately. First of all, some sort of platforming level engine, possibly using tiles, has to be created, in which the player can move, jump, and interact with different elements like children or level transitions. Furthermore, a system to store and display dialogue is needed. Additionally, some user interface elements like a knowledge selection screen and a main menu have to be created. Lastly, a requirement from the client was that the game was playable in a browser, which means that it has to be compiled for a flash target, thus restricting some options like sqlite databases.

Based on these requirements, we started by creating a basic platforming engine based on the TiledLevel demo provided by flixel, implementing the .tmx file format from the TiledLevel level editor. For this a player class and the TiledLevel class were created, as well as the PlayState already included with the basic flixel setup. Now it was possible to load levels from a file and play them, so the level design work could start. The level was encapsulated in that class, in order to enable easy switching between different levels, which has worked out as intended. After implementing this functionality, the focus was set on the dialogue system, which would be a core system in the game. For this we implemented a XML backend that is loaded using the haxe.xml.Fast library and then displayed on screen. All the logic for the dialogue progression was implemented in the DialogueBox class, which, in hindsight, was not the best decision, since it made some things more difficult and made the class bigger than it should be. This overloading of classes should be avoided, and would be one thing we would definitely want to change in the future. Lastly, the rest of the user interface was created, including a screen to select pieces of knowledge for a child, and a map.

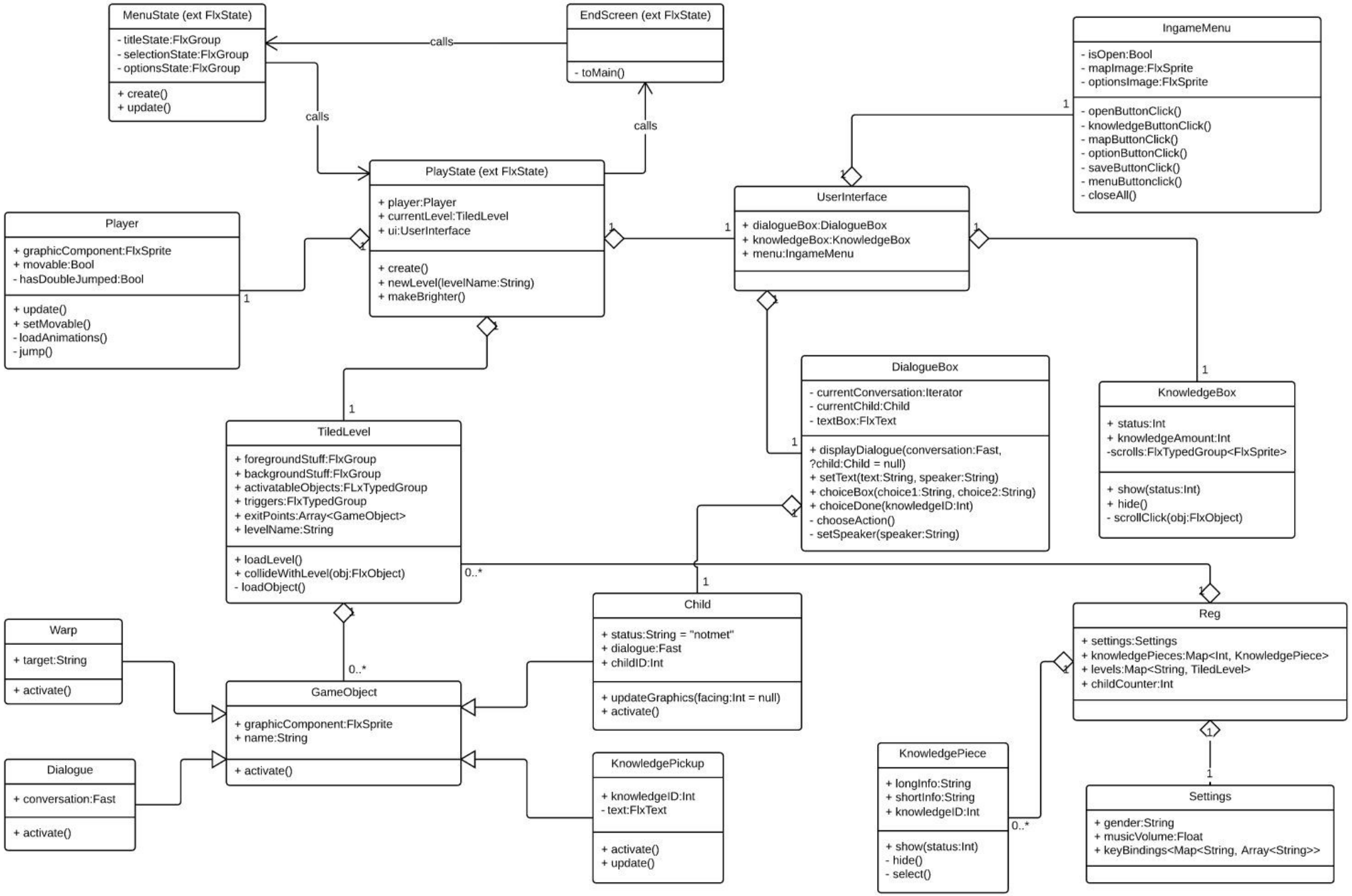
Flowchart



This Flowchart shows the basic progress of the player through the game. He starts in the main menu/title screen where he can choose to openthe options menu. From there on he proceeds through the character selection screen to the introduction of the story and the controls in the overworld, where he can then interact with the first child. Afterwards he goes to the first level and collects the knowledge piece which can then be used to help the child. This repeats for all children in the game until the player has helped everybody, at which point the game automatically ends. At any moment while playing in a level or the overworld the player can also open the in-game menu and choose from the options "Knowledge", "Map", "Options", and "Quit".

Class Diagram

This class diagram (see next page) shows all classes used in our game. The main game is created in the PlayState, but most of the logic is in the other classes. The movement and input management are in the player class, the tilemaps and objects are in the TiledLevel class, and the user interface is in the UserInterface class. Most of the references to things like the KnowledgePieces and the levels are kept in the registry class for easier access, and also to make for easier implementation of a save feature in the future. The registry also contains the settings file, in which all the settings that the player should be able to change in the future are stored. At the moment only the volume settings can be changed from the main menu. The knowledge pieces are created by the KnowledgePickup class when activated and then immediatly stored in the registry, since the KnowledgePickup will get destroyed shortly after.


MenuState (ext FlxState)
- titleState:FlxGroup
- selectionState:FlxGroup
- optionsState:FlxGroup
+ create()
+ update()
calls
EndScreen (ext FlxState)
- toMain()
calls
calls
IngameMenu
- isOpen:Bool
- mapImage:FlxSprite
- optionsImage:FlxSprite
- openButtonClick()
- knowledgeButtonClick()
- mapButtonClick()
- optionButtonClick()
- saveButtonClick()
- menuButtonclick()
- closeAll()
PlayState (ext FlxState)
+ player:Player
+ currentLevel:TiledLevel
+ ui:UserInterface
+ create()
+ newLevel(levelName:String)
+ makeBrighter()
UserInterface
+ dialogueBox:DialogueBox
+ knowledgeBox:KnowledgeBox
+ menu:IngameMenu
Player
+ graphicComponent:FlxSprite
+ movable:Bool
- hasDoubleJumped:Bool
+ update()
+ setMovable()
- loadAnimations()
- jump()
DialogueBox
- currentConversation:Iterator
- currentChild:Child
- textBox:FlxText
+ displayDialogue(conversation:Fast, ?child:Child = null)
+ setText(text:String, speaker:String)
+ choiceBox(choice1:String, choice2:String)
+ choiceDone(knowledgeID:Int)
- chooseAction()
- setSpeaker(speaker:String)
KnowledgeBox
+ status:Int
+ knowledgeAmount:Int
-scrolls:FlxTypedGroup<FlxSprite>
+ show(status:Int)
+ hide()
- scrollClick(obj:FlxObject)
TiledLevel
+ foregroundStuff:FlxGroup
+ backgroundStuff:FlxGroup
+ activatableObjects:FLxTypedGroup
+ triggers:FlxTypedGroup
+ exitPoints:Array<GameObject>
+ levelName:String
+ loadLevel()
+ collideWithLevel(obj:FlxObject)
- loadObject()
Child
+ status:String = "notmet"
+ dialogue:Fast
+ childID:Int
+ updateGraphics(facing:Int = null)
+ activate()
Reg
+ settings:Settings
+ knowledgePieces:Map<Int, KnowledgePiece>
+ levels:Map<String, TiledLevel>
+ childCounter:Int
Warp
+ target:String
+ activate()
GameObject
+ graphicComponent:FlxSprite
+ name:String
+ activate()
Dialogue
+ conversation:Fast
+ activate()
KnowledgePickup
+ knowledgeID:Int
- text:FlxText
+ activate()
+ update()
KnowledgePiece
+ longInfo:String
+ shortInfo:String
+ knowledgeID:Int
+ show(status:Int)
- hide()
- select()
Settings
+ gender:String
+ musicVolume:Float
+ keyBindings<Map<String, Array<String>>