Assignment 2 - documentation

For assignment 2, we developed a multi-level-loader, multiple gamestates, powerups, graphics settings, background audio, *and* a store. This was a lot of work for one week, but we've done our best with the time given. Some implementations might still be a little bare-bones though.

- The store works as follows: after a level is completed, the store opens and by clicking on a power-up, the power-up can be applied to the player. When the player presses the return button on his keyboard, the game goes on to the next level.
- Power-ups don't have a visual representation yet, but they do affect gameplay.
- The wallet does not reset after a game is restarted.
- Multiple levels are not yet loaded from a file, but can be soon.
- We did manage to crank up testing to above 50%.
- RDD was used throughout.

Requirements for assignment 2

Exercise 1 *- The user is able to navigate through different screens. The first is the menu, and the other ones are different levels of the game.*

Must

- When the player opens the game, a menu should open up.
- In the menu, the player can navigate to the main game.
- The menu is a separate state from the main game content.
- In the game, when a player completes a level, he moves to the next level.
- Each level has a different design.
- Each consecutive level has increased difficulty.

Should

- If the player wins the game, a win screen shows up.
- If the player loses the game, a lose screen shows up.
- The player can navigate from the win or lose screen to the menu.
- Other levels have new 'building blocks', like platforms and moving walls.

Could

- The player can navigate from the main menu to the settings menu.
- There are platforms that move around. When the player stands on this platform, he moves too.
- In the settings menu, the player can adjust screen options and audio options.

Would

- In the settings menu, the player can set level difficulty.
- The menu utilizes animations and fancy graphics.

Exercise 1 *- User is able to have a better game experience with audio and better visuals.*

Must

- The game produces a song on opening.
- The screen size must be adjustable in the following 16:9 native resolutions:
    - 1280x720
    - 1366x768
    - 1600x900
    - 1920x1080
- Anti-Aliasing can be switched off with a hard-coded boolean.
- The game can be run in full-screen.

Should

- The game produces sound on GameEvents, like a rope shooting, collisions, bubble bouncing.
- The game utilizes Scalable Vector Graphics (SVG) for flexibility in changing resolutions.

Could

- The player can change sound settings in the menu.
- The player can change the graphics settings in the menu.

Would

- The audio and songs used in the game are specifically composed for our game.
- The soundtrack is built out of different layers that can be changed on demand of the game. When a player moves closer to a bouble, the ‘dangerous’ soundtrack layer is faded in and blended with the rest of the soundtrack.

Exercise 2 *- The user scores money from points and can spend this money in a shop to gain power-ups.*

Must

- The player is shown a store where he can buy power-ups, after completing a level.
- The player has an inventory where currency and power-ups are stored.
- At least 3 power-ups are available.

Should

- The store has to be visually appealing.
- Power-ups aren't lost every level.

Could

- Power-ups are upgradable.

Would

- DLC power-ups

Responsibility Driven Design

Below are the CRC cards for all the new classes to fulfill these requirements.

Multiple levels

Level	
Superclass(es):	
Subclasses:	
Making a Level consisting of walls and a ceiling	Walls
Send everything to game	Game
Manage a timer	Timer
	Ceiling
	Player
	Bubbles

Video options

GraphicSettings	
Superclass(es):	
Subclasses:	
Handles the different graphics settings	View
	Resolution
	SlickApp
	Model

Resolution	
Superclass(es):	
Subclasses:	
Handles the different resolutions	Graphics
	View

Power-ups

Wallet	
Superclass(es):	
Subclasses:	
Keep track of money	GameView
Observe the state of the bubbles	BubbleEvent
	Observer

PowerUp	
Superclass(es):	
Subclasses:	
Powerup: Give extra time	Timers
Powerup: Give extra health	GameView
Powerup: Slowing bubbles	Observer
	BubbleEvent
	Model
	Player
	Bubble

Multiple states and levels

SlickApp	
Superclass(es): StateBasedGame	
Subclasses:	
Set up gamecontainer and states	GameView
	MenuView

GameView	
Superclass(es): BasicGameState	
Subclasses:	
Run the game	Model
	View
	Controller

MenuView	
Superclass(es):	
Subclasses:	
Display the menu	
Move to the next view	StateBasedGame

ShopView	
Superclass(es):	
Subclasses:	
Show the shop	GameView
Handle shop events	GameView

The ShopView now has too many responsibilities; this will be fixed in a later version.

RoomIterator	
Superclass(es):	
Subclasses:	
Handles the loading of Room objects	Room

UML

At the end of the sprint, we have made the following UML diagram. While creating the UML diagram, one things stood out: some parts of the model are called from outside of the Model class, while it would be good to keep those all in the one class. We hope to fix this in the next sprint. A vector version of the uml diagram is to be found in the uml folder in the documentation folder, called “uml_assignment2.eps”.