**SNAKE CAVE**

**DOCUMENTATION**

**What is Snake Cave?**  
Snake Cave is like the classic Snake, where you control a snake that grows longer with every piece of “food” you eat (in this game it’s called candy), but with a random generated border. The goal of this border is to make the game more dynamic and challenging. Every playthrough is different and maybe even easier/more difficult. The player can also choose between three difficulties in the options menu: Easy, Normal and Hard. Easy has a big field, slower snake and more candies. Hard has a small field, fast snake and a small amount of candies.

The game is made in Unity3D.

**The Level Generator**  
The most important part of this game is the LevelGenerator script. It’s what makes this game unique. How this works is it starts with checking what difficulty the player chose. On basis of that it sets the min and max screen sizes and places the LevelGenerator object an the min X and max Y position. Then it calls an IEnumerator that generates the border.

The IEnumeratoe has a do while loop that declares a random float between 0 and 1, which is the direction and calls two functions: CallMoveGenerator(int randomDirection) and CreateTile() (which instantiates a tile object on each position the generator comes if there’s not already another tile and puts them in a list) while the iterations are less than the amount of tiles, which is a public variable.

CallMoveGenerator(int randomDirection) calls in its turn the CreateCorner() function, that changes the direction chances of the LevelGenerator at each corner position and checks when the generator is at its end, and the MoveGenerator(int direction) function on base of the chances the gameObject has to go up, left, down or right.

MoveGenerator(int direction) moves the transform.position of the gameObject with a switch statement. 0 = up, 1 = right, 2 = down, and 3 = left.

Finally the SetValues() function is called from CreateCorner() when the LevelGenerator is at its end position. This function sets the min and max values of alle the created tiles (that’s what the list is for). This is used as min and max positions for the candies to spawn between.

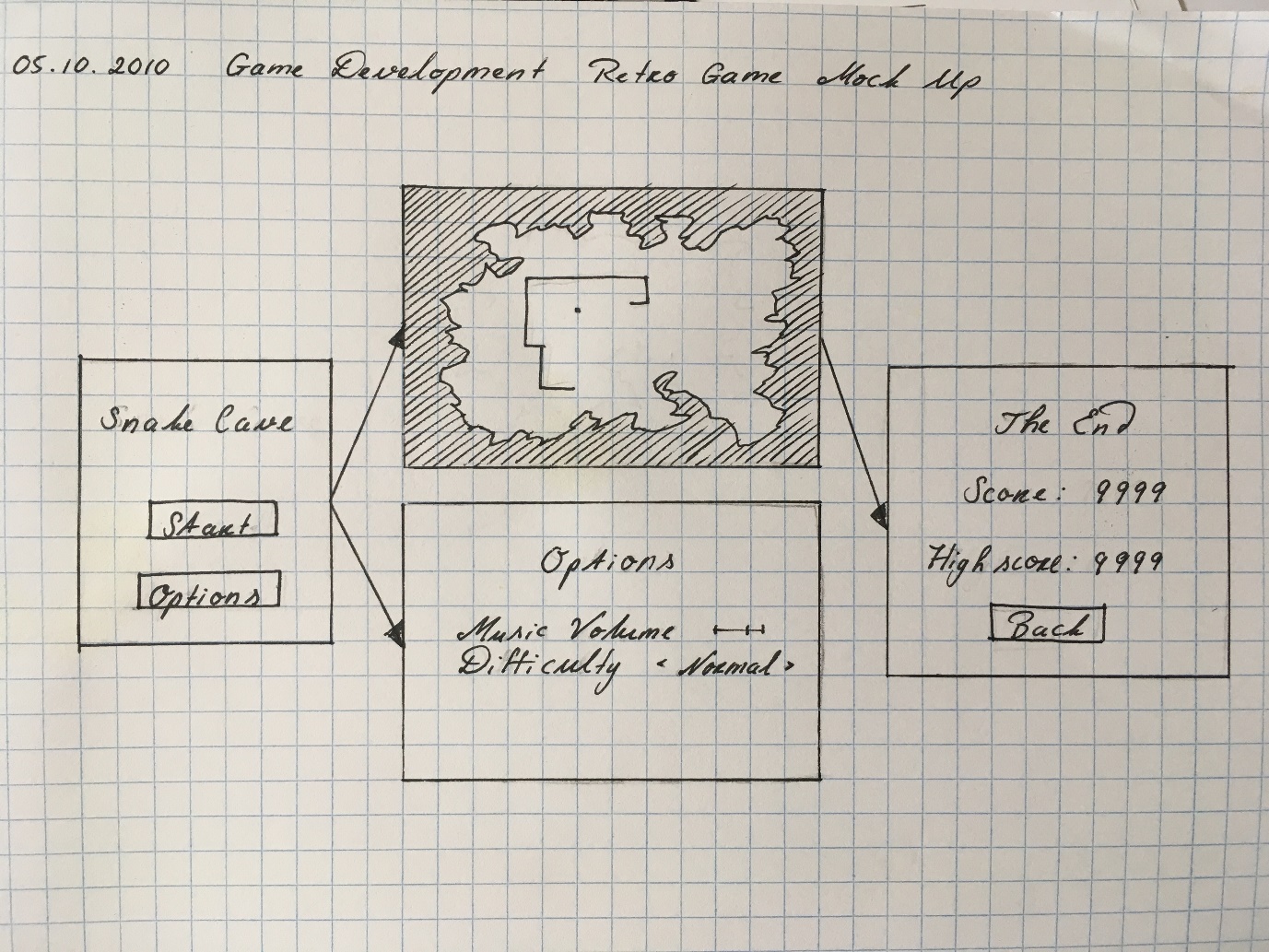
**Process**At first I made the LevelGenerator with four different objects at each corner of the field. These objects then each went its own direction, which eventually formed a playfield with four lines. There are several reasons I deleted this idea.

Firstly and most importantly, the objects didn’t always touch each other’s tiles, which resulted in one or more gaps when the generator was done. This was unacceptable.

Secondly, when the generator did touch the tiles of the next generator, it didn’t stop, but continued. It resulted it some ugly and unnecessary lines.

That’s why I chose to use only one object from which I change the directions at each corn­er.

**Mockup:**

****  
*I’m aware the date in the photo says 05-10-2010. I think we all know how wrong that is.*

**Features\*:**(P1) Game Scene

* (P1) User-controlled snake
* (P1) Random generated walls
* (P1) Candy objects spawning on a random spot between the walls
* (P1) Snake gets longer when picking up candy
* (P1) Collision detection between snake and wall, and snake and tail
* (P2) Score display
* (P3) Sound effects
* (P3) Baby snakes inside a small cave you can feed to get shorter // not in the final build

(P2) Start Scene

* (P2) Play button
* (P2) Quit button
* (P3) Options button

(P3) Options Scene

* (P3) Music volume slider
* (P3) Difficulty

(P2) End Scene

* (P2) Score
* (P2) Back button
* (P3) High score
* (P3) Quit button

\*  
(P1) top priority  
(P2) middle priority  
(P3) lowest priority

**Design Patterns:**

Object Pool  
I use object pooling for the candy objects in the game. These objects appear on the screen, disappear when the snake eats them and appear on a random place elsewhere. I could do this by instantiating and destroying them each time, but what is way more efficient is using an object pool.

In an Awake function I instantiate the number of candies required for the difficulty, set them inactive and put them in a list. When the LevelGenerator is done drawing the borders, I position them on a random place between the borders and set them active. When the snake eats one, I set that one inactive, and iterate through the list with a for-loop. I place the ones that are inactive on a new random position and set them active).

Doing this I only have to instantiate the candy objects once and I never have to destroy them.

Singleton  
I have a music player in the game. This music player can have one and only one object. Otherwise the objects would play simultaneously, which would result in one big mess. That’s why I use the Singleton Pattern. I create a static variable of the MusicPlayer and call it instance. I set this variable initially to null. In the Start() function I check if the scene already has an instance. If this is the case, I destroy the MusicPlayer object. If it’s not, I assign instance to ‘this’ (MusicPlayer object) and call DontDestroyOnLoad(gameObject), so that it transfers between different scenes.