X-terminate - A Simple Side-Scrolling Game.

Jarl Silvén (921027-2739) och Simon Hellberg (940903 8636)

December 11, 2016

Objective and Requirements.

The purpose of this project was initially to record and playback sound through the ChipKit board, however, a busy autumn made us postpone acquiring the necessary equipment. When time ran out, and realizing that the sound quality did not turn out as expected, we switched to a simple 2D game, using text symbols from the labs as graphics. Many features were achieved in the end:

- A moving "side-scroller" screen where rows of characters move to the left based on a timer.
- A player character, controlled with two buttons for moving upwards and downwards.
- Random generation of non-player characters, spreading out obstacles, power-ups and treasure.
- A score counter, displayed at the end and based on distance traveled and treasure picked up.
- Hit detection, allowing player to pick up power-ups and ending the game if an obstacle.
- God mode, initiated when picking up power-ups and allowing the player to pass through obstacles 10 steps.
- Fade out mode, warning the player that god mode is fading by flicking between god character and player character.
- Increase in speed as time goes by, to make the game more challenging.

What we did not achieve due to lack of time but would've liked to was:

- A high score page, stored between games and saved when the ChipKit board shuts down.
- Replacing the rows of text symbols with pixels, and creating more interesting character entities.

Solution.

We developed our game using the ChipKIT Uno32 board and the Basic I/O shield. Most of the code was written in C. We used the buttons to control the movement in the game and the timer to control the pacing. To generate our "random" numbers we used a predefined library, seeded by a combination of user input and the timer.

Verification.

We had to test all the parts of the game and how they interacted with each other. The buttons, the random generation, the power-ups, the hit detection, the game moving faster and the start screen.

Luckily due to the simplicity of the game most of these were simple to test and verify by playing the game, trying to verify correctness in the general case then try to achieve specific niche situations in the game. To test the niche situations we sometimes changed modifiers within the game. For instance to test the power ups we increased the likelihood of power-ups appearing (to test if they interfered with each other or themselves) and to test the winning end screen we made the game easy to complete.

Contributions.

Simon focused on the hit detection and generating a random algorithm that made the difficulty of the game be in line with our vision.

Jarl focused on making the buttons work in the way we intended (making sure every button press only moved the player once etcetera) and designing and implementing the power-ups. We wrote the rest of the code together.

Reflections.

In the end, we both regret not starting on the project sooner, as we underestimated how difficult input and output would be through the Basic I/O Shield. Learning more about input and output would have been very rewarding, but practicing C code had great value as well.

More than anything else, the project gives one an appreciation for high-level programming languages. Though C is more efficient and closer to hardware, other languages supply better structure and libraries allowing one to focus on designing cool features rather than reinventing the wheel and making rudimentary connections to input and output.

Next time we come across a project such as this, we'll make sure to learn the IO better and keep that in C, but switch to a high-level language as soon as it gets too complex.