Legend of LCOM

Class 4 - Group 8

Francisco Teixeira Lopes – 201106912

Rui Miguel Almeida Oliveira – 201000619

Description

Proof of concept clone of The Legend of Zelda based on the NES/FC version. The game has several maps that can be traversed by exiting through the edge of the screen, enemies move around in these maps trying to deal contact damage to the player. Player can interact with the mouse or keyboard to move the character or attack. Initial goal of the game is to survive, later versions would have an actual objective like the real game.

Target grade: 20

Devices

Timer

Role: Framerate control for the main game loop

Functionality: Square wave mode with interrupts

Keyboard

Role: Game input for interacting with the player character and menus

Functionality: Scancode reading with interrupts

Video Card

Role: Draws all game objects, enemies, game world, menus, etc

Functionality: Ability to set modes and to alter the VRAM

Mouse

Role: Alternate control scheme

Functionality: Mouse packets with interrupts

RTC

Role: Timed challenges, puzzles or countdowns with alarm functionality

Serial Port

Role: Person in another PC can use the keyboard (possibly mouse) to spawn enemies in-game

Modules

Main – main interrupt handler, functions as game loop. Responsible for updating the display and game state. Similar to a Dispatcher.

Developer: Francisco Lopes

Program Logic – updates game state, checks collision, parses inputs and changes the Game World accordingly.

Developer: Francisco Lopes

Game World – keeps the world state in static structs (e.g array of structs containing enemies), other modules interface with this one when they need to know about game entities, map information, etc. Can return members of the structs if called by other modules, akin to getter methods in C++.

Developer: Francisco Lopes

Timer – timer functions used for the game loop, contains utility function timer_sleep. Most of this is already done from lab2.

Developer: Rui Oliveira

Keyboard – keyboard functions to return make/break codes, lab3 is pretty much this. These codes are later handled in Program Logic to alter the game state (e.g move character, attack, navigate menus).

Developer: Rui Oliveira

Mouse – mouse functions to return mouse packets, button states and movement details, these are parsed later in the Program Logic.

Developer: Francisco Lopes

Scancode – handles Keyboard scancodes to convert them to ASCII, useful for remapping controls since the user shouldn't have to look at unparsed codes, low priority module.

Developer: Rui Oliveira

Graphics – handles every graphical aspect of the game, the idea is to use a VGA mode that allows 32bbp (true color) in Direct Color Mode with a Linear Framebuffer. A later iteration would work in different resolutions but is not a priority. A graphics submodule loads image files to use as sprites or maps for the game, this will use an existing library, specific library still to be decided.

Developer: Francisco Lopes

RTC – handles alarms for events (e.g timed puzzles, countdowns) and also returns current date if needed.

Developer: Rui Oliveira

Serial Port – handles sending/receiving bytes through the serial port. Current suggestion is using another keyboard to spawn enemies in the game so this module would just send a scancode to the other PC and the other PC reads the UART registers to check if information was sent.

Developer: Francisco Lopes

Menu – simple options menu when starting the game, also handles a pause menu when in-game.

Developer: Francisco Lopes

Optional Modules

These modules are very low priority and won't be implement until the rest of the game is perfected, consider this more of a wish list. As such these modules do not have a developer in charge.

Save/Load – if the game ever gets big enough to warrant a save/load game option this would be the module for it. Since we're cloning The Legend of Zelda, it would make sense to have this since the original game was the 1st console game to ever implement a save game feature.

Level Editor – ideally the game map would be loaded from files with some kind of ASCII to game object encoding. A level editor would have a GUI that converted these game objects back to the ASCII encoding so the game could load them. Imagine a format similar to CSV files, each line would include an ASCII for each "tile" in the map.

Plan

1st week – 32bpp VGA mode working with importing of image files using a library.

Module: Graphics

2nd week – basic game loop with no collision but with player interaction for moving the char.

Module: Keyboard / Scancode

3rd week – try to have collision implemented, at least with enemies if not the game world.

Module: Timer / Program Logic

4th week – try to implement mouse controls and RTC.

Module: Mouse / RTC

5th week – extra stuff and serial port, also useful to catchup if project is late.

Module: previous modules if late / Serial Port / Menu / Save-Load