Castellum

Game Development Final Project - Intro to Programming

David Arppe, Damian Lal, Ben Kupka, Kyle Blumreisinger, Donald Smith
Studio 16, KB3D
University of Ontario Institute of Technology

The game Castellum is a top-down maze game, created using Visual Studio 2010 and the Public Domain Curses library. The game focuses on puzzles as the main gameplay, rather than violence, and this broadens the target audience to all ages. It has replay value, when the player chooses the "Maze Race" mode in the menu, and it has a main story where the player races to the top of a tower, to prove herself worthy of travel outside the village. Castellum works off of an engine that can be adapted easily to add more, and converted from ASCII graphics to 2-Dimensional graphics. The game imports the maps used in the story mode from text files, and the sprites are loaded from the ASCII art file.

I. Introduction

Castellum relies on the PDCurses library, a public domain curses library for DOS, OS/2, Win32, X11 and SDL. It implements most of the functions that were available in curses, which is why it was chosen for the program. The game Castellum is an ASCII game, with ASCII characters that represent sprites in 2-Dimensional space. The curses functions for windows and placing characters local to coordinates were ideal for our program, because it is not like most ASCII games. We tried our hardest to implement graphics like that of SNES games, despite our limitations. Each "Sprite" consists of 54 characters, in a 9x6 grid. These appear relatively square on the monitor, because ASCII characters are taller than they are thick. The code is across 9 files, but a function on the top of the tree can still, and often does, return an integer value. This value is passed on through the returns of up to 4 functions before returning to the MainGameLoop.cpp, where the loop is either continued, or exited. This is one method we have used. The other, is used in changing the games state. The games state is one of 3 things. The first is the drawn background that prompts the user to press any key. The second is the menu, with its sub-menu's. The third is the over world. When a game is started, the game state is changed by changing the variable gameState to 2. Our code also implements pointers. This was an essential part of our game.

II. Detailed Description of Program

The program starts in the MainGameLoop.cpp, and it defines the variables it will use. Some of these are loopGame, and gameState (19) which are used only inside the MainGameLoop file. Inside the main function (40) is a while loop (60) with a switch statement (67) inside of it. The while is the main game loop. The game will only loop so long as loopGame is true. This is how functions will exit the game. They return a variable, and loopGame is set to that integer they return. As long as it is one, it will loop. On line 70, you can see an example of the loopGame getting its value. This one in particular is from the int DrawBackground(char*,int) function, in DrawWindows.cpp on line 280. Inside of the function, the executable reads the file passed on from the user, and draws it as the background. We return 0 (306) if the file doesn't exist. This value gets returned to the MainGameLoop.cpp. Inside the game loop, we have the switch statement (67). This determines what the game will be looping. If the gameState is -1 (69) then the game will be looping, and waiting for any key to be pressed. If the gameState is 0 (73) then the game draws the Main Menu, and the main menu will determine what happens next.

The int MainMenu() function is inside of the DrawWindows.cpp on line 49. This is where the player first decides what game they want to play, whether it be the infinite randomly-generated maze mode offered, or the story. Exiting the game is also an option. On line 62 inside DrawWindows.cpp, there is a switch statement. This creates an interactive menu, where the user can see which option is highlighted before actually selecting the option. This is the part of the interactive menu that changes where the selection is. Public Domain Curses allows us to call getch(), and it returns an integer value, based on the character pressed. It also defines some integers when you initialize it. KEY_UP (64) is one of the integers that has been defined. It is the integer that getch() would return when you press the up key. These change the variable selection (30), which is the option the player wants to press. Our Menu is limited to 4 options, though if more are needed, the programmer can write a 4th option possibly by the title "More Options" that will bring a new set of options. The 4 options are constant, and are stored in arrays of pointers to characters (36, 40, 44).

WINDOW *create_win(int height, int width, int starty, int startx) (83) is the function used to create windows. The order the coordinates are written in PDCurses is Y,X, so we tried to follow it as best as we could in our functions as well. This function creates a local window (85), to store the information for the window that will be returned (91). A new window with the specifications is created (86) and then we remove the

border by creating a custom blank border (89) then the executable calls wnoutrefresh(local_win) (90) which doesn't refresh the window, but adds it to a list of windows that need to be refreshed. This greatly improves performance, because once every frame, by simply calling doupdate() (DrawWindows.cpp 311, RoomController.cpp 70). It is only called twice in our code, which makes it easy to manage what is drawn and when. Windows are drawn in the order they are added.

In the MainGameLoop, RoomUpdate() is the function called in the overworld (79) Room Update is part of its own .cpp file. RoomController.cpp was created for Room Update, and other functions that will control the rooms. Importing the ASCII art is also inside the file. It will return 0 if the file can't import, and this will go back to the MainGameLoop, and stop the loop. RoomUpdate() will start a nested for loop, these cycle through every object in the currentLevel. currentLevel is a class that holds all of the objects in the games current room. puzzleObject and roomObject are both objects, which was defined in Object.h. What the nested for loop does for the roomObject is animate the sprites, like water. What it does for the puzzleObject is check to see if movement needs to be calculated (Boulders are the main puzzle objects). RoomUpdate() also has a part that calls the PlayerMove() function. This is located inside the PlayerMovement.cpp, and uses a different method of checking whether a key is down. The second method of checking doesn't use getch(), but checks the flag for the key.

III. Programming Concepts

The following are some of the concepts discussed in class that were used in the project along with where they can be found in the code. First off, the project heavily utilizes classes, as more or less everything that appears on the screen is of class Object. This Object class is defined within the files Object.cpp and Object.h, the class can be viewed on line 7 of the header file Object.h (Appendix page 74). A second large dependency in the project is file I/O, files are used to read in all the maps in the game, as well as the sprite sheet which has all the sprites in the game. The sprites in the game are assigned to objects, and the maps are made up of these objects. The maps are drawn through a switch statement which can be seen on from line 32 to line 1022 of the file CreateObject.cpp (Appendix page 5). The sprite sheet is read in before all the sprites are assigned to objects, that is, it is read in when the main game loop is initiated, and can be found within the main function on line 52 of MainGameLoop.cpp (Appendix page 3). The maps are then read in via the ImportMap function found on line 33 of the file ImportMaps.cpp (Appendix page 37). The switch statement in CreateObject.cpp is important, but it is not the only vital switch statement used in the program. Another switch statement is used when dealing with the interaction of the player with other objects in the game. This switch statement can be found from line 22 to line 552 of the file Interactions.cpp (Appendix page 41). Pointers are also used frequently within the program when dealing with objects, windows, and arrays of characters. Pointers are used when passing objects through functions, an example of this can be seen within the PlayerMove function on line 22 of PlayerMovement.cpp (Appendix page 62). Another use of pointers is when we are creating windows, which can be seen on line 24 of DrawWindows.cpp (Appendix page 27) or on line 83 of DrawWindows.cpp (Appendix page 28). Lastly pointers are used to have arrays of characters in place of strings, examples of these can also be found in DrawWindows.cpp on lines such as 33, 36, 40, and 44 (Appendix page 27).