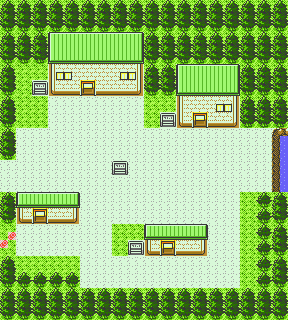
UG Mini Project - RPG V1.0 (Working Title)

Overview

For my mini project I developed an RPG style game, I use the term game loosely as there is not much of a story line to the game. Unfortunately when I started I didn’t realise what an undertaking this would be from a programming perspective in terms of the amount of time it would take and the difficulties I would run into. Thus I have tried to do what I can programmatically at the detriment of a story line. The program therefore works more so as a proof of concepts rather than a full function game with a story line.

In terms of the game, it is an old-school tile-based RPG, in the vein of older Final Fantasy or Pokémon games, potentially too much so. The image here shows the idea behind this.



(Pokémon Gold - http://cdn.wikimg.net/strategywiki/images/6/6b/Pokemon\_GSC\_NewBarkTown.png)

The reason I chose to do this was that as a teenager I was quite engrossed with these games and had a general idea how they worked having toyed with ROMs and map editors for such games; which gave me the basis for using arrays to store the map data. Once I had the basis for the maps, calculations for the rows and columns to extract from the sprite sheet, set up I was also able to create a Town Editor for the game also. So that if I had more time I would have easily been able to create plenty of maps and areas within the game.

The game’s sprites came from OpenGameArt (<http://opengameart.org>) and some of the ideas have come from <http://www.tonypa.pri.ee/tbw/start.html>, although the site is talking about making a tile based game using Flash. As such all of the code has been done from scratch and not taken from another source, although the structure is based on the above website’s suggestions.

How to Use

The mouse is used to click on things like menu items, i.e. the new game and load game options at start up and in game menu items. The player is controlled using the arrow keys, the spacebar hides dialogs or moves to the next dialog box. The in game menu is accessed using the Enter key. In battles the number keys are used to select the relevant move to use.

Within the game you should be able to walk up to the animated enemies and battle with them, the houses are entered and a random battle is generated when walking through the green area on the second map , gotten to by walking the player along the road to the bottom of the screen on the initial map.

Arrays – map class

The game’s world is made up of four 3D arrays; maps, moveMap, dialogs and battles. Each of these is used as reference to the relevant information for a particular area, which is made up from four 2D arrays i.e. when accessing area 1 (maps[0]), the 2D array for that map is accessed and similarly, the moveMap, dialogs and battles with the same index (0) will access the 2D array for a particular area at the same index. (There could have been a fifth for overlaid objects, but I just left these out due to time constraints).

Maps[][][]

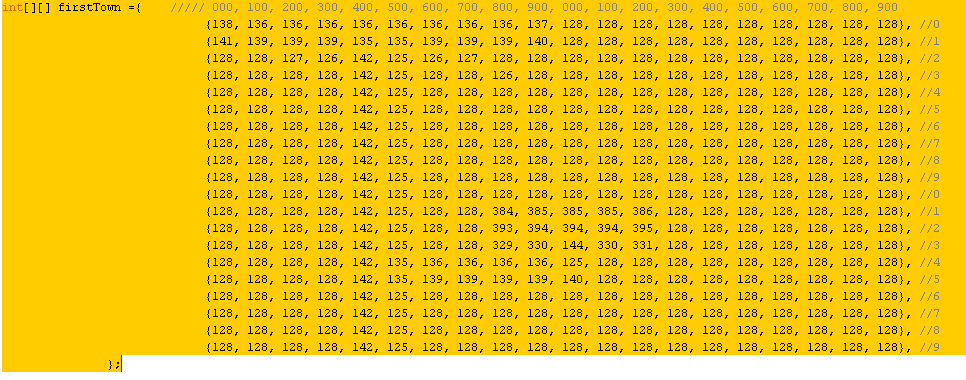
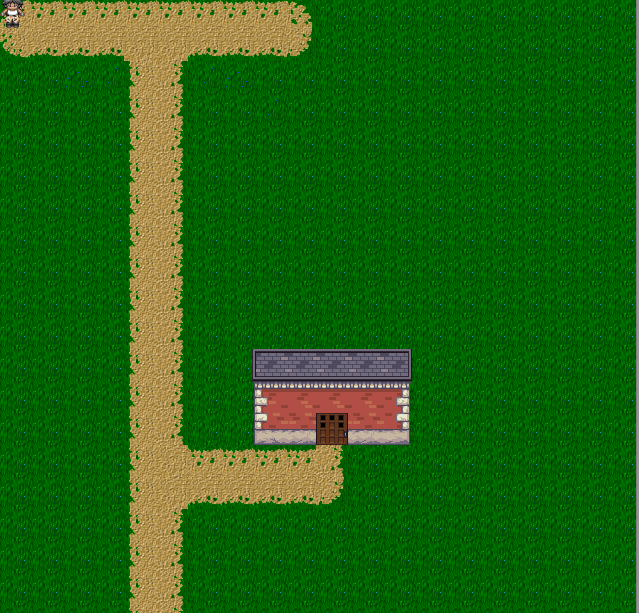
That data for all of the mapped areas are stored as 2D arrays. Each with its own particular variable name that is then stored within maps[][][]. This allows for each area of the word to be referenced by an index.

The 2D array for each map contains integers which are used to calculate which area of a sprite sheet should be textured to a 32x32 pixel tile’s shape. For example, an integer of 0 would refer to the first 32 pixels across and down, 1 would refer to the square of pixels from 32 to 64 in the x-direction and 32 in the Y direction, based on the tilesheet being 2048 pixels across i.e 64 tiles. Below is an extracted part of the sprite sheet showing this, the sheet in reality continues on for several rows and columns.



Part of spritesheet (sprite2.png)

This results in the following lines of code, producing the proceeding image on screen:

Source codeGame Screenshot

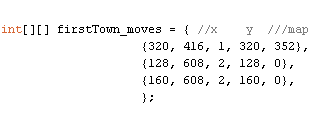
All maps are then “collected” using the following line of code

and it is this array that is accessed when loading an particular area’s map.

moveMap[][][]

The moveMap[][][] array is used to store, 2D moveMap arrays for the locations within an area that will result in the player moving to another map, in reality the movement is done by setting the required currentMap variable to the new maps index in the maps[][][] and then the player’s position x and y to whatever the desired position is.

Each moveMap[][] is structured as follows:



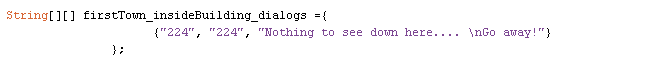
The first and second element state the positions x and y which will result in the map changing, the third element is which map to change to, and the fourth and fifth elements say which co-ordinates that player should be at when moved.

Similar to the 3D array structure of the town maps, the moveMaps are collected into a 3D array and then are accessible using the same currentMap variable used to access the world map.



dialogs[][][]

The dialog arrays store the x and y positions within an area that will trigger a dialog box, and the resultant text to display.



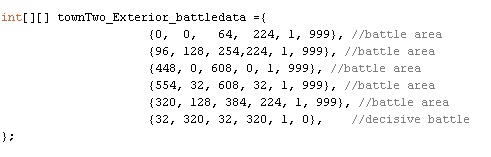
The data is stored as strings, however the x and y coordinates are then cast as integers when checking if the player’s position x and y match up.

Again these are then collected into the 3D array dialogs[][][] and are referenced using the currentMap variable.



battles[][][]

The battle information stored in a 2D array is structured in such a way that battles can be decisive i.e. a particular enemy is set at a particular position. Alternatively they can be random within a particular area.



The code above is structured such that the first two elements are an x and y coordinate, as are the third and fourth, within which the player will encounter a battle. However the first five arrays in the example above are square areas within which a battle will occur and the fifth is for a particular tile at which an enemy is located. In this particular case the sixth element states which enemy is located there 999 for random and 0 for the first enemy in the enemies array. The fifth element states whether the enemy is visible or not i.e. in decisive battles has the enemy been defeated.



Again all of these 2D arrays are put into a 3D array and then accessed the currentMaps variable.

Building The Map – mapBuilder class

The mapBuilder class is used to structure the data taken from the arrays belonging to the map class. This includes taking the the integers from the map array and creating till objects from it, this includes information as to where the tile is and whether it is walkable or not. A tile is walkable if its reference integer is below 185. This is a point within the sprite sheet where the majority of the tiles are not walkable. As I didn’t create the sprite sheet myself, some may look like they should be however had the graphics been done by me I could have decided more precisely and positioned the sprites to my liking.

The functions of the mapBuilder class include dialogTrigger, battleTrigger, switchMap, drawEnemyTiles, drawMap. The mapBuilder constructor is what sets up the tiles and then the drawMap function will then use the tile.display() function of the tile class to draw them. The tile class uses the function doSpriteRow to calculate the positions of the sprite sheet to texture to the square vertices.

The dialogTrigger and battleTrigger functions are relatively similar in functionality. Each one contains a FOR loop, which iterates through the dialog and battle 2D arrays based on the current map variable and if the player’s x and y position are within these areas then the event is triggered, whether that be a dialogbox or a battle. The battle function is more complex however as it has to determine whether to trigger a specific battle or to randomly generate a battle with a random enemy.

Battles

The battle system in place is a turn based system. The programming for this involved a large amount of if states and variables which keep track of whose go it is, player or enemy, and the dialog for each part of the battle.

Unfortunately I did not have time to program the battle system to have different use a properly system to decide which moves have what effect, which ones are less effective etc, so currently every move will remove five heath points from the opposition.

Also each battle takes in the player’s moves and the relevant damage points, this was to allow these variables to be changed by items used by the enemy or player without affecting the real states of these things outside of the battle, however again the programming for this was not done.

Menu

The in-game menu class is an inherited dialog box, with additional functions for checking which menu item has been selected, each of which returns a Boolean and functions for the rollover effect when the mouse is over a menu item. The menu can be used to display the player’s in-game stats and save/load the game.

Saving the Game

Initially I wanted to save the output for the game as XML however this proved to be slightly complicated therefore the game data is saved to a text file as required and then can be loaded from the text file also. Due to time constraints of programming I have only programmed it to save player information such as position, map, HP and level, but it could also be set up to save which enemies have been defeated, etc and then load this data also.

To do the saving I have used the createReader (<http://processing.org/reference/createReader_.html>) and createWriter (<http://processing.org/reference/createWriter_.html>) functions.

These allow variables to be output to a text file as a string, when the file is then read upon loading a save game the string is delimited by a comma and converted into an array, the in-game variables are then set using the array’s elements.

Town Builder

As an addition to the game I have made a town builder, in terms of functionality it only allows for building the actual map, but realistically to be fully function would be extended to include adding in dialogs, objects, changes between maps etc.

The program initially has a 2D array the same size as would be used by a map in the game with all elements set to 999 which will give an empty unwalkable tile within the game. The user can then select which tile they want to use from the sprites at the top of the program and then by clicking on the grid place the tile.

The town building program reuses the doSpriteRow method from the RPG to display the tiles within the grid. The program uses calculations of the mouseX and mouseY position when the mouse is clicked over the sprite sheet image to decide which tile the user wants to select. When the use then clicks within the grid the relevant tile number is placed into an array and the tile displayed. If ‘S’ is pressed the entire array is then output to a text file so it can be copied into the RPG game.



Town Builder in use