Mini project

RESEARCH ASSIGNMENT

201321527-Brendon Clark

Unable to find Plagiarism cover sheet

Brendon Clark
5-8-2016

Introduction

I have decided to develop a game as part of my mini project. The project makes use of various data structures. The game is a 2d platformer with enemies and items. Items can be picked up and dropped through the game. The game idea follows a number of steps namely:

Step 1 - You play as the main individual player.

Step 2 - You are required to move the player through the level and pick up 3 items while fighting off enemies.

Step 3 – After you collect all the 3 items you need to find a door which will check if you have the items in order. If the items are in order you win the game. If you do not have the items in order, you will need to rearrange them into another order by dropping the items and picking them up again. Alternatively, you will need to pick up other items as you proceed.

The objective of the game is to follow a specific order. If the order is incorrect or not systematic, you will be unable to proceed to win the game. There can be multiple number of arrangements however, the first element must be a 'battery.'

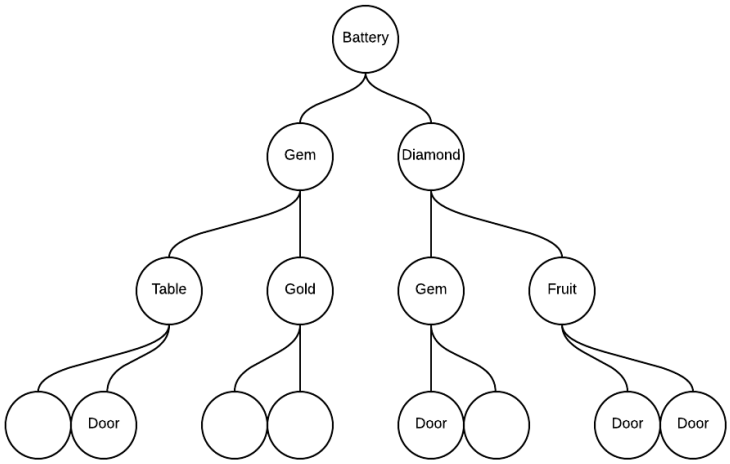
Data structures

Based on the development of the game four data structures were used as namely Stack; Queue; Binary Tree and Arraylist as described below:

Stack - There are 5 stacks used. They are used for positions of the game objects for the enemies and the items. There reason it was appropriate to use a stack for this is because the order of the positions of the items and enemies do not need to be in order and you can also get O(1) insertions and removals. This is ideal because when an item or enemy is instantiated the position stack can be used to set positions of the items and enemies. This is the only usage for stacks in my implementation of the game.

Queue – There is one queue used which is responsible for the storing of items that you pick up as you play the game. This is identified as the "bag" of the player. It is appropriate to use this data structure because you can get O(1) insertions and deletions. The added benefit of using a queue is that all items that you pick up will be in a logical understandable order, and when you drop items the logical understandable order is still experienced.

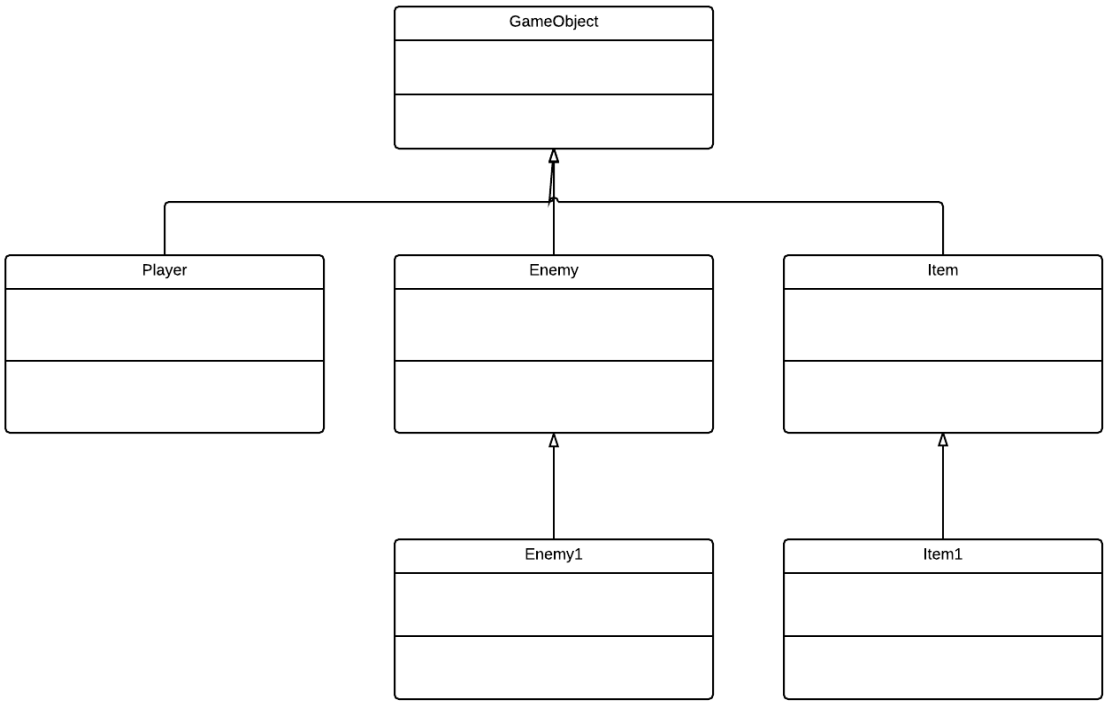
Binary tree – The binary tree is used for the main logic of the game. The tree is instantiated with pre-defined elements. These can be several different items in the tree. The player must pick up items which will go down the tree, until you reach a ‘door’ on the lowest level of the tree. On the lowest level there are only four (4) doors which the player can reach by picking up items. If you get to a door you win and if not, you do not win. The tree is as follows:



ArrayList – The array list from java was used for the storing of enemy objects as well as the item objects. An array list was used because it provides O(1) insertions. Furthermore, it is to my knowledge that java’s array lists have been modified to be made as fast as possible. An added bonus was the ability to use the for each loop for iterating through enemies and items.

Architecture

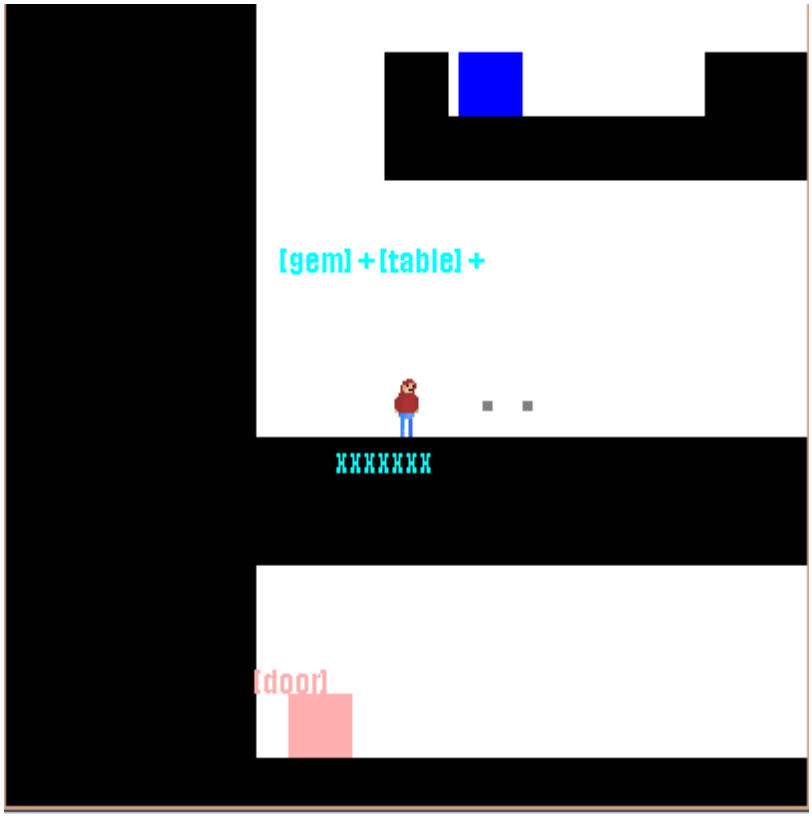
The architecture I used is based heavily on inheritance in order to use polymorphism and abstract implementation on base classes. Another reason inheritance was key is to provide generality for certain relatable objects. The general architecture of a game object is depicted below

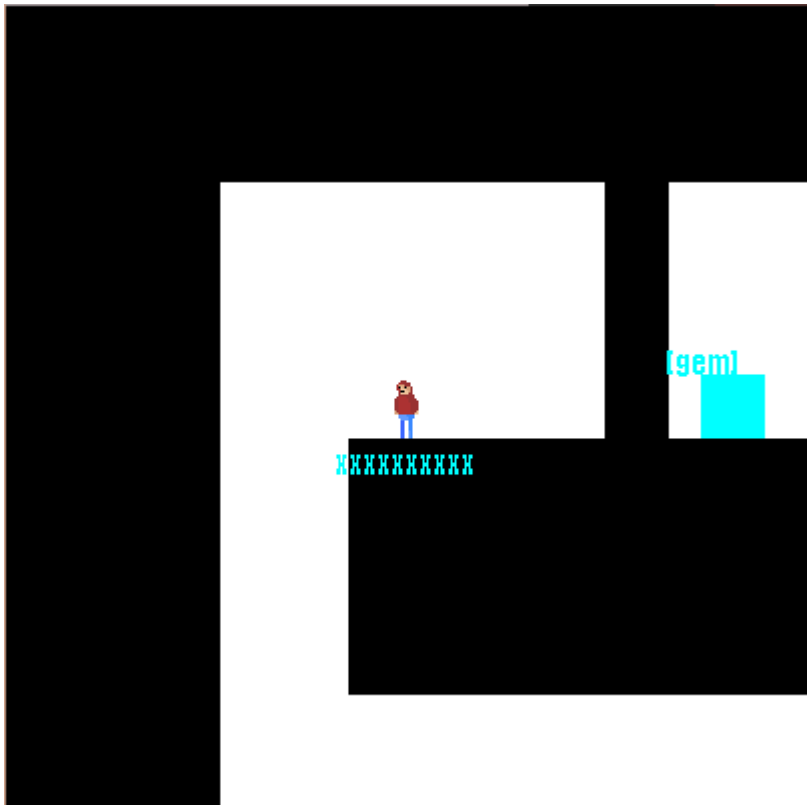


From the image above, it is easy to see the main architecture of the game where all objects inherit the super class GameObject. This allows for the low level implementation to be done in an abstract super class GameObject. The GameObject is the class needed for all game objects of the game such as enemies, players and all items. Within each base class derived from GameObject class there are various other objects that use composition to achieve the desired result. From using this architecture, the implementation of various objects became much easier and more focused for each object.

Screenshots

Player shooting and has picked up 2 items





Player standing life points at the bottom of player.

Game engine

The game engine architecture is from a book called *Killer game programming* (Davison, 2005). This book highlights a very good method for game development and graphics handling in java. The game engine I developed, makes use of a technique called double buffering which helps to paint graphics onto a frame/panel. This technique ensures that there is little flicker and tearing of graphical objects and provides smooth graphics handling.

Game state manager

The game state manager object allows for a game to have multiple game states. My game only has 3 states which are the menu, the game and the paused state. The game state manager allows for an interaction between the various states. The game state manager ultimately allows for more control over a game as a whole.

Trees, stacks and Queues

All tree, stacks and queues where developed by myself and referenced from the prescribed textbook on the course namely, (Goodrich, Tamassia, & Goldwasser, 2014).

All trees and stacks where implemented in separate packages in order to achieve modularity within the game implementation. The stack and queue data structure where only used for their O(1) runtime properties. The stack and queue where not iterated. Because of this, it allowed for faster runtimes. The tree data structure did not make use of tree transversal techniques however it was iterated through with a custom algorithm to provide the core point to the game. This custom algorithm achieved O(n) runtime which is not ideal however, there are not a lot of elements to iterate through.

References

(n.d.). Retrieved from OpenGameArt: opengameart.org

(2015). Retrieved from StackOverFlow: www.stackoverflow.com

Davison, A. (2005). *Killer Game Programming in Java.* O'REILLY.

ForeignGuyMike (Director). (n.d.). *Java 2D platformer (youtube)* [Motion Picture].

Goodrich, M. T., Tamassia, R., & Goldwasser, M. (2014). *Data Structures & algorithms.* Wiley.