CSC591 (Game Engine Foundations) Homework 4

Homework Spec

The homework spec can be found in `[root_dir]\docs\HW4.pdf`. Below is a brief excerpt from the same:

Your task for the fourth and final assignment is to implement a script management system, some scripted functionality, and a new game (or games) to demonstrate the resuability of your engine design.

Introduction

My assignments have been building on one other. And by that respect, I probably had to make the least amount of change for this assignment. I simply added the script manager and added respective events to handle them. Even for the two games that I built, I almost did little to no change to my engine. All this will be apparent as I go through the following sections. Unlike before, I will be presenting steps to run my program individually in each of the sections, instead of one set of instructions at the very top. This is because we would be needing at least three different executables for this assignment.

Scripting

Running my program

There are two ways to run my multiplayer game / engine demo with scripting enabled:

1. **JAR**:
    1. Find the JAR file for this project in `[root_dir]\out\artifacts\multiplayer_game_engine_jar\multiplayer-game-engine.jar`.
    2. Open a command line and type `java -jar multiplayer-game-engine.jar s` (for server).
    3. For running clients, open other command lines, type and execute `java -jar multiplayer-game-engine.jar c` as many times you want for any number of clients.
    4. Remember that you need the run the server first and then the clients, otherwise this might throw some exception. This should be normal, as for most multiplayer games, the headless server generally is always running.
    5. Also currently, the client searches for a running server in `localhost`, so running the server and client in different computers will not work. If you still want to run it in different computers, follow my second way of running the program and before building it, open `Constants.java` and assign the server's IP to the `SERVER_ADDRESS` String variable.
    6. The scripts will be present in `[root_dir]\out\artifacts\multiplayer_game_engine_jar\scripts\` folder. If you want to change the script, you can open `script.js` and modify accordingly.
2. **IntelliJ**:
    1. Install IntelliJ Community Edition.
    2. Import and build my project.
    3. There should be two run configurations - one for the server and one for the client. Run the "Server" first and then the "Client". The shortcut for running programs in IntelliJ is `Alt + Shift + F10`.
    4. If you don't find the run configurations, make two yourself. For the sever, give a command line argument of `s` and for the client, give a command line argument of `c` (without the quotes).
    5. The scripts will be present in `[root_dir]\scripts\` folder. If you want to change the script, you can open `script.js` and modify accordingly. Remember that this folder is different that what was shown above. This is because I have to keep the scripts folder relative to the executable. So if I have two executables (the JAR and one run through IntelliJ), the paths will be different and there has to be different scripts as well. Moreover, the JARs don't pack the scripts, so they have to be separately supplied to them.

Explanation and features

There are two ways of handling scripts in my game - one being the vanilla way of running a script in every update cycle. This is showcased by the "Modifying game objects" sub-section below. In this case, the `controller` class registers a set of bindings (of objects) with the engine which it wants the scripts to have. And then in every update cycle, based on the toggle script value, the script is executed. The function to be executed is also registered with the engine. The other method of calling

scripts is on-demand. And this is showcased in the “Handling events” sub-section. Here, I only execute scripts when some specific event(s) arise. The significant parts of my scripting are written below:

1. **Toggling the script**: In-game, you can toggle execution of the script by pressing the `T` key. I have kept this because I don’t want to run every script from the moment the game starts. I want to find the right time to run it. Also, I think giving this power to the game developer is invaluable. All these instructions are present on the bottom-left corner of the screen.
2. **Modifying game objects**: The function `stairMover` does this. It allows changing the positioning of the standing stairs (static rectangles). There are at max three static rectangles in my game, so I have three blocks for moving the stairs up, down, left or right. If you see any statement, for e.g. `standingStair2.moveUp(0)` - the function name describes what I am trying to do here. In this case, I am trying to move the stair up with (currently) `0` velocity. So it will not move at all. You can change this to something else and see the stair move, without the need of any re-compilation. This is an example of modifying game objects (specifically the position in this case) in the game.
3. **Handling events**: My user inputs are driven through events (well almost everything in my game is driven by events now). I have modified my game/playground such that, when the scripts are toggled, the user input is handled through scripts. This is done in the `handlePlayer` function. When the user input event is raised, it goes to my event handler and based on the script toggle, it raises another event for the script. This again comes to my event handler class, but in a different method specifically reserved for scripts, and then the user input event is passed to the script through the values `key` (which key was pressed) and `set` (whether it was pressed or released). The way that I have handled user events in my scripts has a subtle difference. Previously pressing `A` or `D` would move the player left or right. And pressing `SPACE` would make the player jump. But now, pressing `A` or `D` would make the player move and jump at the same time. There’s no way to just move left or right any more. You can see this in my creencast, when I toggle the script and move my players, they will never move just left or right, but will always have a jumping component to their movement. This is not necessarily anything that improves a game, but instead is just a way to showcase that I am handling user inputs through scripts now.

Thoughts

Implementing the scripting was rather easy. I built over the code that Professor had given in class. The only non-trivial part was setting up the execution of the script correctly, so that all the objects to be bound and the function to be called gets properly passed through. As I am trying to build an engine which can be plugged in to build any game, scripting is an important part of it. So scripting will have to be a part of my engine. So I had to build ways so that my game can properly pass important information about script paths, bindings and script function names and then let the engine handle the execution in the right moment. In that my current game can be replaced with another game, and no change needs to be done at the engine level.

Screencast

I have uploaded a screencast to YouTube so that it’s easier for you to check what I’ve done. The demo shows two clients and shows the scripts being toggled and values changed to immediately reflect the same in-game. First, I show the game objects being modified and then I show player movement after toggling a script. It might be a little difficult to understand when exactly am I toggling the script, because that is not reflected on-screen, but you can understand that it has been toggled when you see my character not making any straight left or right moves, but always jumping. And then toggled back to not using scripts when it starts making those kind of moves.

https://www.youtube.com/watch?v=3hwwboad5Js&feature=youtu.be

Space Invaders

Introduction

I have made the Space Invaders game as my second game. It uses a client-server architecture and is a single player game. I will give instructions on how to run the program and in the following sections explain certain parts of the game.

Running my program

There are two ways to run my Space Invaders game with scripting enabled:

1. **JAR**:
    1. Find the JAR file for this project in `[root_dir]\out\artifacts\space_invaders_jar\space-invaders.jar`.
    2. Open a command line and type `java -jar space-invaders.jar s` (for server).
    3. For running clients, open other command lines, type and execute `java -jar space-invaders.jar c` ONCE. This game is implemented using a client-server architecture but is a single-player game. So there can be only one client. Running more clients may have undefined results.
    4. Remember that you need the run the server first and then the client, otherwise this might throw some exception. This should be normal, as for most multiplayer games, the headless server generally is always running.
    5. Also currently, the client searches for a running server in `localhost`, so running the server and client in different computers will not work. If you still want to run it in different computers, follow my second way of running the program and before building it, open `Constants.java` and assign the server’s IP to the `SERVER_ADDRESS` String variable.

2. **IntelliJ**:

    1. Install IntelliJ Community Edition.
    2. Import and build my project.
    3. There should be two run configurations - one for the server and one for the client. Run the “Server” first and then the “Client”. The shortcut for running programs in IntelliJ is `Alt + Shift + F10`.
    4. If you don’t find the run configurations, make two yourself. For the sever, give a command line argument of `s` and for the client, give a command line argument of `c` (without the quotes).

Controls

The controls are displayed on screen, but still I will give a description of the same here:

1. Press `A` to move left and `D` to move right.
2. Press `SPACE` to shoot bullets.

The game

This follows the classic game rules. I will list them and any variations here:

1. There are 9 rows and 9 columns of enemies starting from the top. In total there are 81 enemies to start with.
2. There are no additional enemies added in the course of the game.
3. The enemies keep going left, coming down, going right and then again coming down, and continues this process. Its velocity keeps increasing with time.
4. Once the player shoots an enemy, it’s color changes to red. And it keeps getting darker if it takes in more bullets, and ultimately disappears when its completely killed. Basically there is a `health` factor for each enemy which gets reduced with incoming bullets.
5. The objective of the game is to shoot all enemies before they can reach the user and collide with it.
6. If the player dies, the game restarts with the number of deaths incremented for the player.
7. A score is maintained as is a live accuracy counter which is based on the number of bullets fired and the ones which actually hit an enemy.
8. The number of deaths reduce the increase rate of the score.
9. The game is continuous, i.e. if you die, then it restarts and you keep building your score.

Client-server architecture

The server here creates the enemy map and sends it over to the client, i.e. it establishes the environment. From there on, the client handles the player movements and killing of enemies.

Scripting

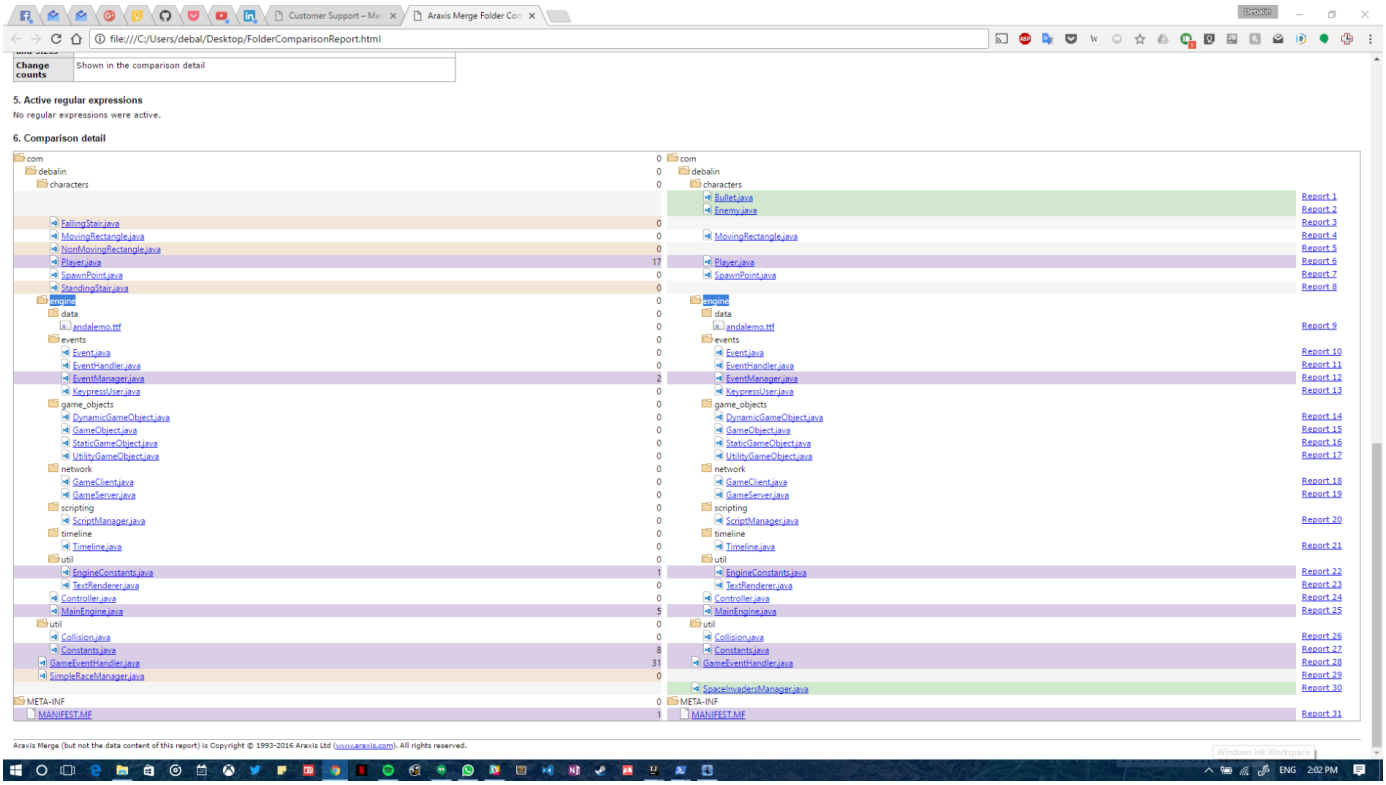
Scripting is enabled in my game and the script file lies in the same directory as my previous assignment section. In the script file I can change the color of my player to something else. I expose the player object to the script and then I call the `changeColor` method in that to change the color.

Difference and code reuse

From the initial assignment, I have tried to make by game engine as pluggable (modular) as possible. Thus when I show the difference report below, you will see that the engine part has hardly changed. I will explain whatever changes were done and it will become clear that the engine was very reusable indeed. Personally, it was incredibly easy to make the game with my engine. That being said, it is also true that I am making a 2D game with one level. With multiple levels, probably some change had to be done to my engine.

I first compared my `src` directory from the `scripting` part of the assignment with the `src` of space-invaders. The former has the latest code for my engine which also the firt game/playground which I was building throughout the semester. While making the second game, I reused the previous engine and removed the game code to add new game code for space-invaders. Our goal is to see how much I could reuse my code from the first game to make my second game.

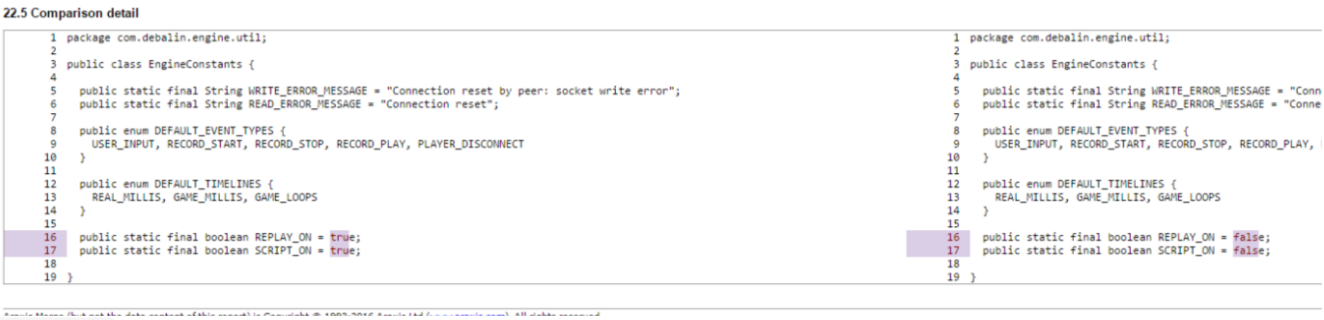
Here is a report for the `src` folder comparison.



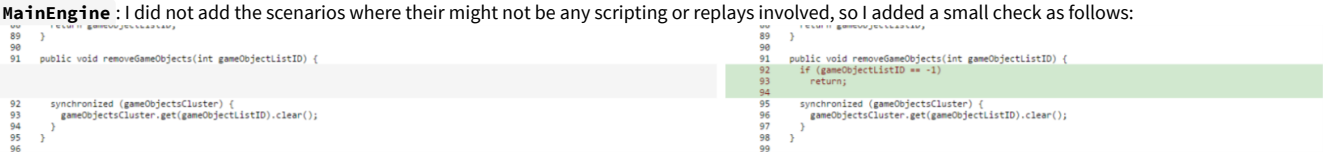
The green lines show new files added, yellow shows removed files and purple shows changed files. You will see many differences here, but it is obvious that as I made a new game the **game code** itself will be different. That shows the files in the `characters` folder, `util` folder and the `GameEventHandler` class, all of which are specifically part of the game that I am building. Of interest to us is the `engine` folder, because that's where all my engine code lies. Let's see what has been changed there. Out of 17 files in engine, only 3 files have been changed for making my second game. They are:

1. **`EventManager`** : The change here is trivial. I had previously used a certain technique to mirror certain events for multiple clients in the recording event queue. In the new file, I removed this completely, as I don't need it anymore thus making it even more reusable.

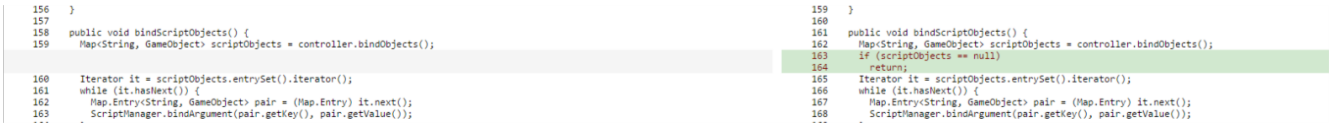
2. **`EngineConstants`** : As I don't need replays here, I have simply disabled it, using a flag. Ideally this should be driven from the game as well, but for this assignment, I kept it like this.



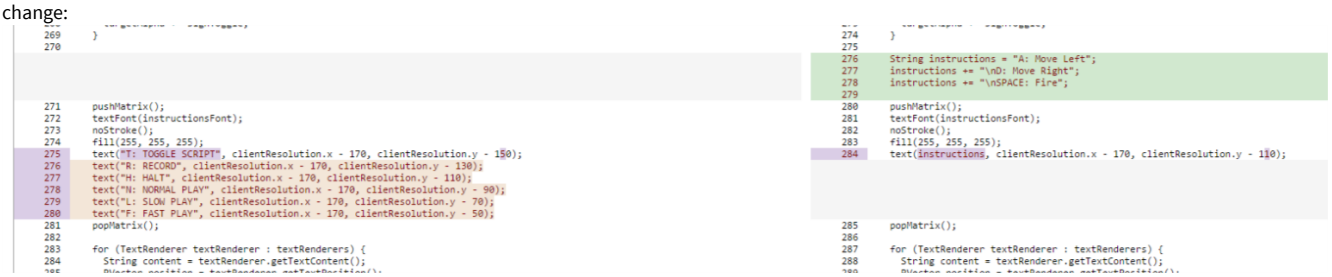
3. **`MainEngine`** : I did not add the scenarios where their might not be any scripting or replays involved, so I added a small check as follows:



This actually makes my engine more reusable from now on. Similarly:



An important change and probably a con of my engine is that I am printing the instructions for playing the game through the engine. This should ideally be handled by a callback registered by the controller and hence the game manager can do that. But due to time shortage, I added it to the engine. And that's this change:



And that's it! All the core game object code, network code, synchronization using CMB (yes, my code still uses CMB), replay, scripting, event management, everything has remained exactly the same as it was for the first game! I can conclude hence that my engine code, is very reusable.

It should also be noted that I am not comparing game code here (engine and game code lie very separately in my implementations). This is because it does not make sense to compare something like `SpaceInvadersManager` and `SimpleRaceManager`. They will obviously be different because they have different characters, different gameplay, rules, etc. So I have only compared the engine part of tmy code to measure resuability.

Full reports are available in the `reports` directory. The reports were taken using a trial version of Araxis Merge.

Thoughts

All in all, it was very interesting to use my game engine and make a new game. And it was very a smooth process, the engine being very reusable. The game itself is a little difficult. I myself have not yet been able to kill all enemies in one go. Hope you like it!

Screencast

I have uploaded a screencast to YouTube so that it's easier for you to check what I've done. I play the game one time and when I die, you can see that my deaths increase and the game restarts. I also show the usage of scripts in the end.

https://www.youtube.com/watch?v=p15XHEM9_8o&feature=youtu.be

Bubble Shooter

Introduction

I have made the Bubble Shooter game as my third game. It uses a client-server architecture and is a single player game. I will give instructions on how to run the program and in the following sections explain certain parts of the game.

Running my program

Same as before, just change `space-invaders` with `bubble-shooter`.

Controls

The controls are displayed on screen, but still I will give a description of the same here:

1. Move the mouse to control the turret direction.
2. Press `SPACE` to shoot bubbles.

The game

This follows the classic game rules. I will list them and any variations here:

1. There are 9 rows of bubbles initially.

2. Additional rows get added from the bubbles that you shoot.
3. The bubbles keep coming down at a constant pace.
4. There are three colors of bubbles: red, green and blue.
5. The objective is to shoot and match all bubbles before it reaches the red line.
6. If the player loses, the game restarts with a new set of bubbles.
7. Three bubbles to be fired are shown beside the player controller turret.
8. A score is maintained which is based on the number of bubbles you match.
9. Two bubbles of the same color are matched. The matching is done with neighbors on the left, right and two on the top.
10. The game is continuous, i.e. if you die, then it restarts and you keep building your score.

Client-server architecture

The server here creates the bubble map and sends it over to the client, i.e. it establishes the environment. From there on, the client handles the player movements and killing of enemies.

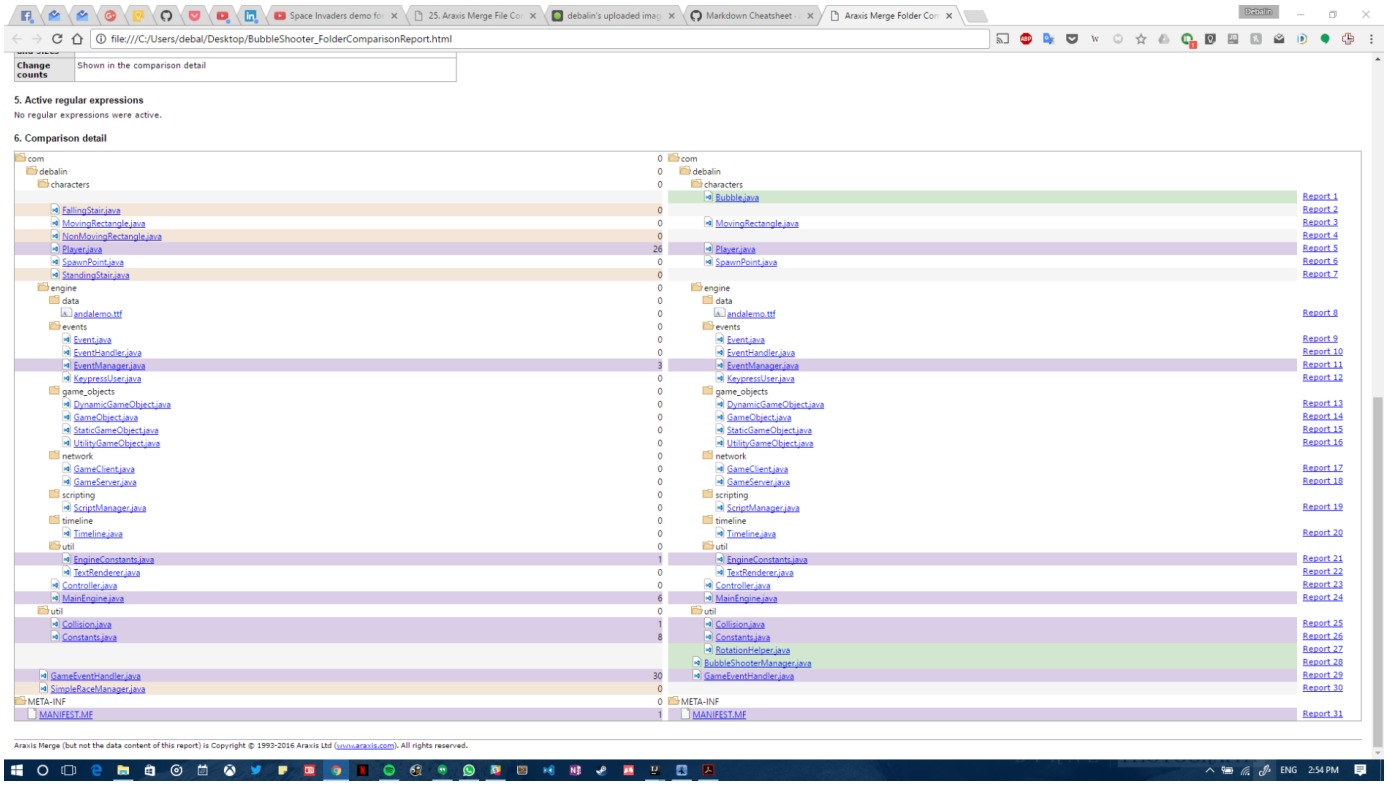
Scripting

Scripting is enabled in my game and the script file lies in the same directory as my previous assignment section. This is the same as the last game's script. In the script file I can change the color of my player to something else. I expose the player object to the script and then I call the `changeColor` method in that to change the color.

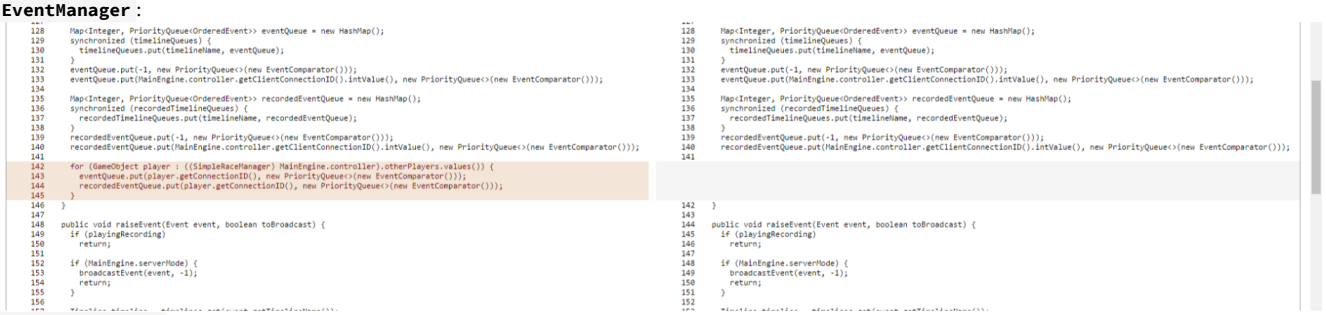
Difference and code reuse

This is exactly the same as my second game report and I have made no extra changes to my engine between my second and third game. So, I will just give the new set of difference images below.

Here is a report for the `src` folder comparison.



1. **`EventManager`** :



2. **`EngineConstants`** :

22.5 Comparison detail

3. **`MainEngine`** :

Full reports are available in the `reports` directory. The reports were taken using a trial version of Araxis Merge.

Thoughts

It was very interesting to use my game engine and make another new game. Like before, it was a very smooth process.

Screencast

I have uploaded a screencast to YouTube so that it's easier for you to check what I've done. I play the game one time and when the bubbles reach the bottom, you can see that the game restarts with the death count increased by one. I also show the use of scripts in the end.

https://www.youtube.com/watch?v=o7P0cNh5ofI&feature=youtu.be

Reflection

I have already reflected a lot on the game engine reuse part throughout this report, mainly in the `Difference and code reuse` parts. That answers the first few questions in the homework specification about how successful I was in reusing the code and what changes I had to make to accomplish the same. Still I will answer all of the questions in brief here.

How successful (or unsuccessful) you were at designing for reuse?

I think I was very successful at designing reuse. The main reason behind it is that I maintained this stance of making this engine as modular as possible from the first assignment. Sometimes, you lax a little and that had shown up in the part where I write the instructions for a game which has been hardcoded in the engine. Though it can be argued that in this case, I have gone out of my way to allow something which was never a part of my assignment. I can easily remove that chunk of code and there won't be any relations to any particular game in my engine. Rest of the core code, as I mentioned before, like game objects, event management, event synchronization and networking hasn't been changed one bit in my new games. They were initially a part of the engine code and still are. It's just like I have used my engine library and only focused on the game logic.

What systems did you have to change to get your second game to work?

This has been specifically answered in the code reuse sections above for individual games. In brief, I had made a few trivial changes to actually make my engine fare in a more generic way and one change involving the instructions which I just talked about while answering the previous question.

How did you change them?

Answered above and in the previous sections for individual games. Specifically, I allowed for affordances to use or not use the game scripting and replay functionalities.

If you were going to start over again to design your game engine again, what would you do differently?

1. One thing that I had mentioned before was that this game engine specifically caters to 2D games. If I had the time, I would have implemented ways to allow 3D games being developed as well.
2. Processing has been taking care of the graphics part for us. Next time, I would have removed that and built a draw/update loop myself with special attention to adding models and textures.
3. We had not touched on audio which would be a nice thing to have.
4. Sending game objects across the network was different that sending events. My engine either supports game objects or supports events. If I would have known that we need to do both from before, I would have designed it in such a way that it can work with both simultaneously, not one or the other.
5. Sending game objects across networks by having controller callbacks was a nice idea that I had implemented, but I feel there might be a more elegant approach to this problem. I would love to explore this area more and see what I can come up with. More specifically, when game objects are received at one end, they are simply relayed to the controller in the most rudimentary fashion. If my engine could do more, like organize which game objects to receive how and have a game object handler, much like the event handler which says if `x` game object is received send it to this function, otherwise send it to this other function, I think that would have been better. In the HW2, my controller on the game side used to do all this, but if the engine can do this, it would be more feature-rich.
6. I would change CMB to have a less heavy protocol.

What would you do the same?

1. The sending and receiving of data across the network that I did using a bag of tasks approach (using monitors in Java to wait and notify when data arrives) was one of the very cool things that I did and I would keep it that way.
2. The game object class was very generic and would be the same with a probably a few more additions to accommodate 3D stuff.
3. Event Handling was incredibly fun to use after I implemented it. When I needed more events, I simply added them to event types, registered them and raised them. I knew that it would somehow go and reach my event handler class which I had registered with my engine. There, I just added a new method for handling that method. This was very useful and I would keep it that way.
4. Replay system was pretty well done, I believed and I would keep it that way.
5. Scripting, though being basic, has a proper structure and I would not change much of it.
6. There are many other subtle things like the way I have maintained a game object cluster, the way games register their game objects (by getting back a cluster ID), etc. which I feel is well-done and I would keep it that way.