CSC 413 Term Project Documentation

Fall 2018

Student name: Ratna K Lama

Student ID: 909-324-382

Class. Section: CSC413.01

GitHub Repository Link

Tank Game*:* https://github.com/csc413-01-fa18/csc413-tankgame-rlama7.git

In the Den Game*:* https://github.com/csc413-01-fa18/csc413-secondgame-rlama7.git

Table of Contents

1 Introduction

The goal of the CSC413 term project was to build two 2D games written in Java. The first game was to implement the Tank Wars game with set of requirements defined by instructor Anthony Souza. For the second game we could select from a list of provide game options or we could choose to build our own. I chose to build my own game based on the tank game.

My objective of the term project was to practice Object Oriented Principle (OOP) along with source code reusability. I attempted to reuse almost all of my source code from the tank game when building In the Den game when possible with Do not Repeat Your code (DRY) strategy.

1.1 Project Overview

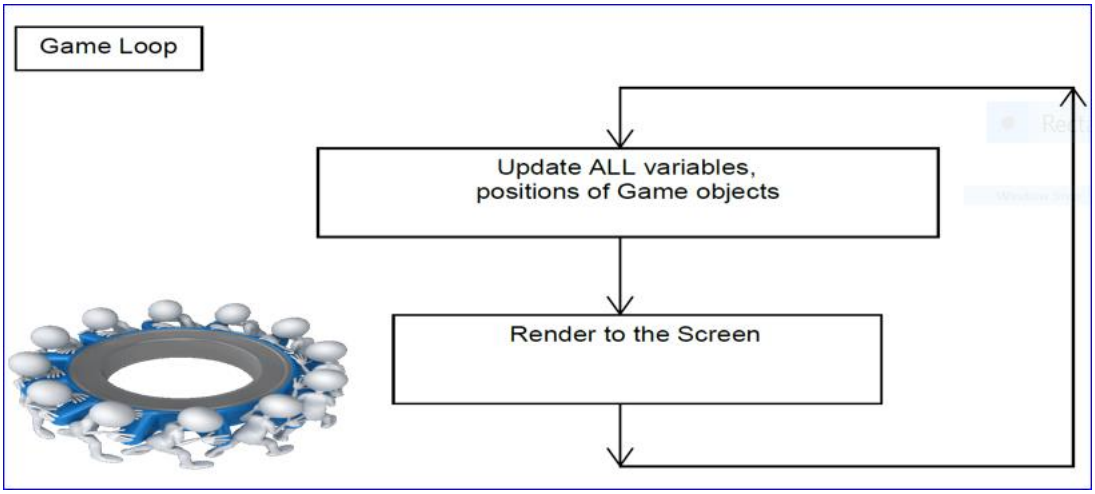
The objective of the term project was to practice good OOP and source code reusability in the process of building 2D game in JAVA. We were required to build two games: Tank War and In the Den game from scratch without using any game engine.

1.2 Technical Overview

JAVA programming language provides rich features to design and build feature rich 2D and 3D games.

C and C++ are de facto language for game developers due to popularity of the Unreal Game Engine. However, there are wildly successful games such as Minecraft written entirely in JAVA. Moreover, Java supports threading internally within the JVM, as such it allows us to take full advantage of multi-core processor to build highly scalable and efficient games.

The heart of both the game is the game loop. Essentially all the variables, graphics and audio are constantly updated with tick() method and rendered with render() method in the run() method in the Game Class.



I implemented runnable interface to run my game entirely on its own thread. Despite many high-resolution graphics and many sound effects, my game performance was smooth. Although I noticed a slight lag in loading the game after adding sound effects, for regular user, the lag should be unnoticeable.

1.3 Summary of Work Completed: Tank Game

1. Two Players
2. Players can move back, forward, left and right direction
3. Item drop feature
4. Breakable Wall (Reddish marble wall)
5. Unbreakable Wall (cement brick wall)
6. Bullet flies in only x-axis
7. Menu option to display player's health and item info to the screen
8. Smooth performance
9. Good OOP practice

1.4 Summary of Work Completed: In the Den Game

1. Single Player
2. Multiple Targets (wild animals)
3. Player can move back, forward, left and right direction
4. Item drop feature
5. Breakable Wall (Red Tree)
6. Unbreakable Wall (Wood Fence)
7. Sound effects
8. Player's health and item info displayed to the screen
9. Graphics and sound effects provide smooth and pleasant game experience
10. Good OOP practice
11. Reused almost all of the source code from the Tank Game

2 Development Environment

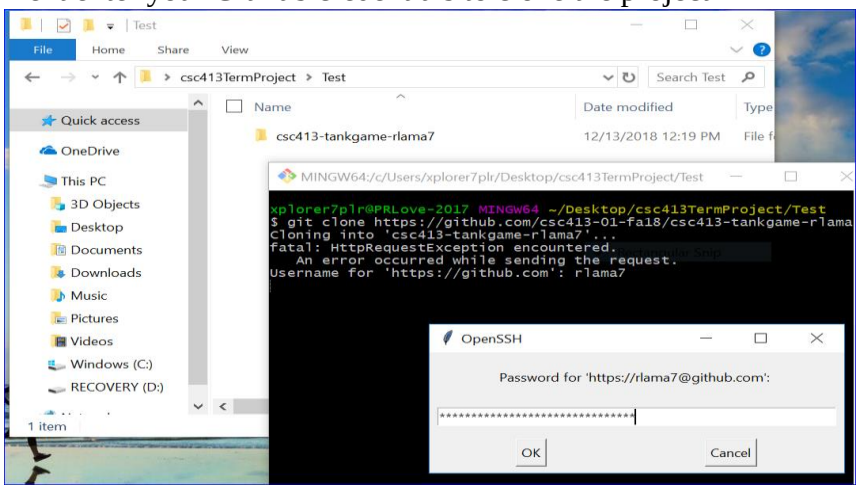
Following development environment made the term project possible:

1. Eclipse IDE Version: 2018-09 (4.9.0); Build id: 20180917-1800
2. Java JDK Version 10.02
3. Git Version 2.14.2 on Windows Operating System (OS) 10
4. UMLet version 14.3, Free UML Tool for fast UML Diagrams
5. IntelliJ IDE plugin: simpleUML version 0.33

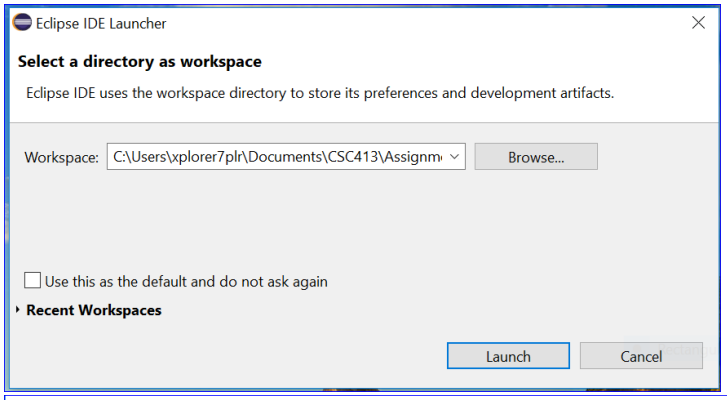
3 How to Clone, Import and Build Project

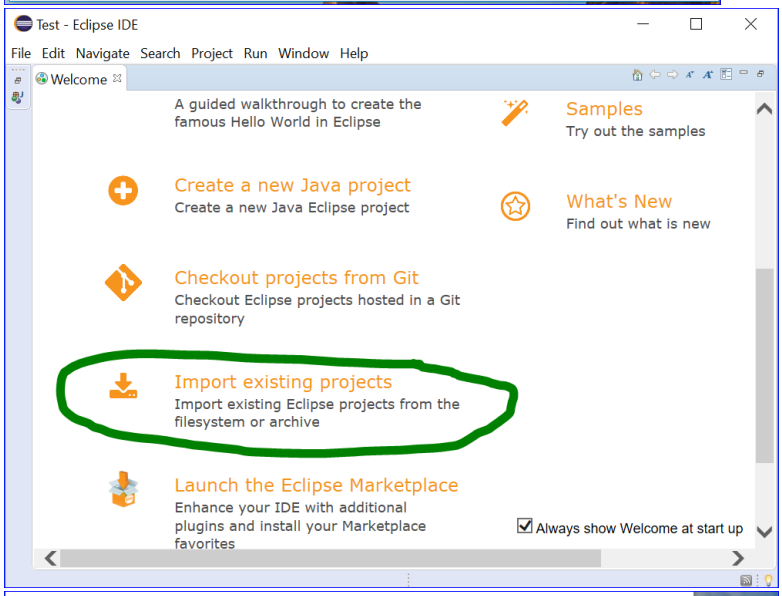
Both the game: Tank Game and InTheDen Game were built using the Eclipse IDE.

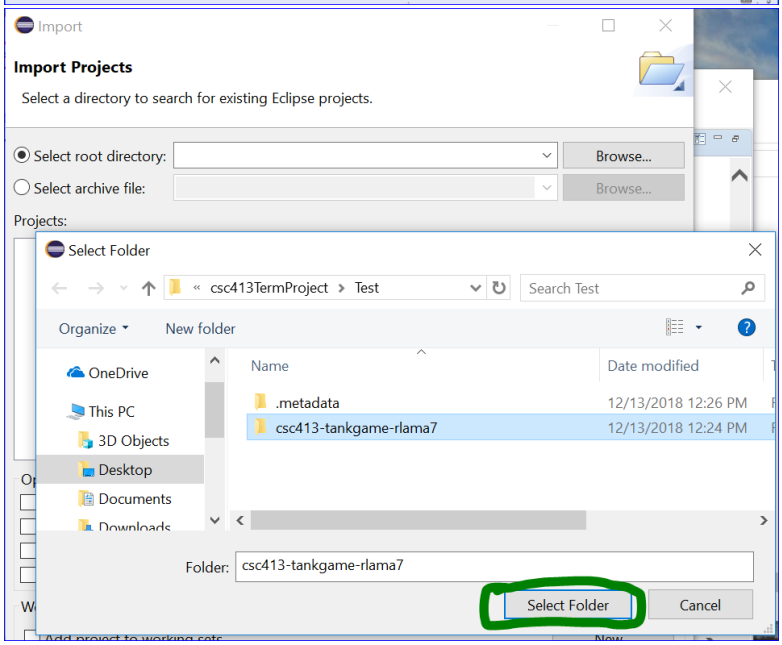
1. Steps to clone project (commands are shown in **bold)**:
    a. Select a folder where you want to clone the project to
    b. Inside the folder right click then select **Git Bash Here**
    c. Using the git bash clone the Tank Game and In the Den Game from the GitHub repository using the following commands:
    git clone https://github.com/csc413-01-fa18/csc413-tankgame-rlama7.git
    git clone https://github.com/csc413-01-fa18/csc413-secondgame-rlama7.git

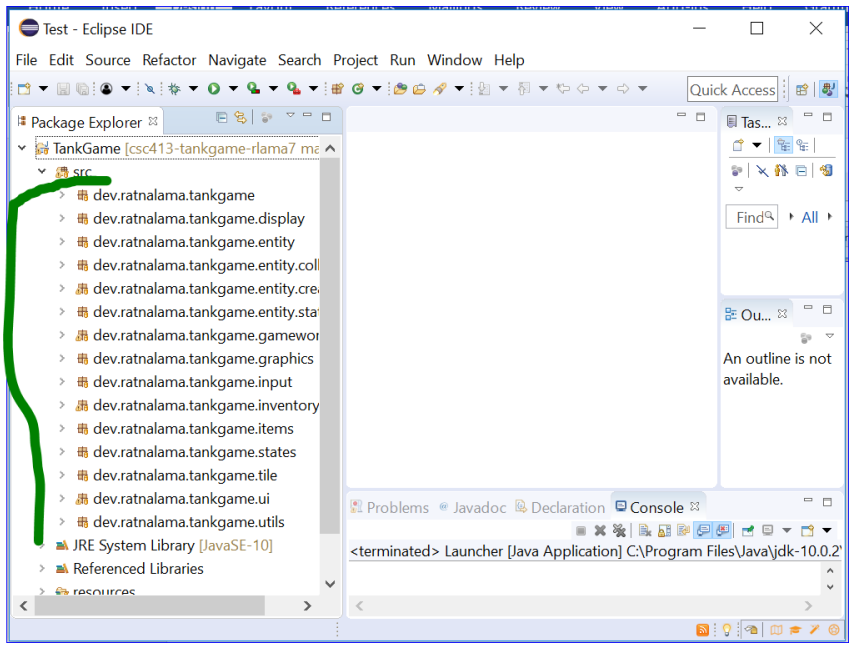
2. Next enter your GitHub credentials to clone the project
3. 

4. Steps to import project Tank Game:
    a. Open Eclipse IDE and select a directory as a workspace.
    b. Navigate to the folder where the project folder was cloned. Inside the project folder, navigate to the folder path: **csc413-tankgame-rlama7**
    c. Next select **Launch**
5. When the Eclipse Opens for the first time, in the welcome page, select: **Import existing projects**
6. Next navigate to the root directory: **csc413-tankgame-rlama7**
7. Next select folder. It should launch the project.
8. In the Eclipse IDE under the TankGame
    a. **src** folder contains all the source code
    b. **resources** folder contains all the resources such as fonts, and graphics.
9. To run the game, select **Run** from the top menu bar

10. 
Eclipse IDE Launcher
Select a directory as workspace
Eclipse IDE uses the workspace directory to store its preferences and development artifacts.
Workspace:
C:\Users\xplorer7plr\Documents\CSC413\Assignm
Browse...
Use this as the default and do not ask again
Recent Workspaces
Launch
Cancel

11. 
Test - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Welcome
A guided walkthrough to create the famous Hello World in Eclipse
Samples
Try out the samples
Create a new Java project
Create a new Java Eclipse project
What's New
Find out what is new
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Import existing projects
Import existing Eclipse projects from the filesystem or archive
Launch the Eclipse Marketplace
Enhance your IDE with additional plugins and install your Marketplace favorites
Always show Welcome at start up

12. 
Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
Browse...
Select archive file:
Browse...
Projects:
Select Folder
csc413TermProject › Test
Search Test
Organize
New folder
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Name
Date modified
.metadata
12/13/2018 12:26 PM
csc413-tankgame-rlama7
12/13/2018 12:24 PM
Folder:
csc413-tankgame-rlama7
Select Folder
Cancel

13. 

14. Follow similar steps to launch second game: In the Den Game

4 How to Run Project from JAR file

4.1 Steps to Run Tank Game from the JAR file

1. Navigate to the folder where the project folder was cloned.
2. Inside the project folder, navigate to the folder: **csc413-tankgame-rlama7**
3. Locate **jar** folder inside folder **csc413-tankgame-rlama7**
4. The **jar** folder should consist of folders:
    a. resources
    b. TankGame (JAR executable file)
5. Click TankGame (JAR executable file) to launch the game
6. **Note**: resources folder and the TankGame (JAR executable file) should always be located in the same folder to launch the game properly

4.2 Steps to Run In the Den Game from the Jar File

From step 4.1 follow steps 1-6 to run In the Den Game from the JAR file.

5 Assumption Made

My assumptions about both the game were the requirements set forth by instructor in his documentation pdf.

6 Class Diagram

Both the Tank Game and the In the Den game shared a lot of classes. In particular, the second game, In the Den reused almost all the source codes from the first Tank Game.

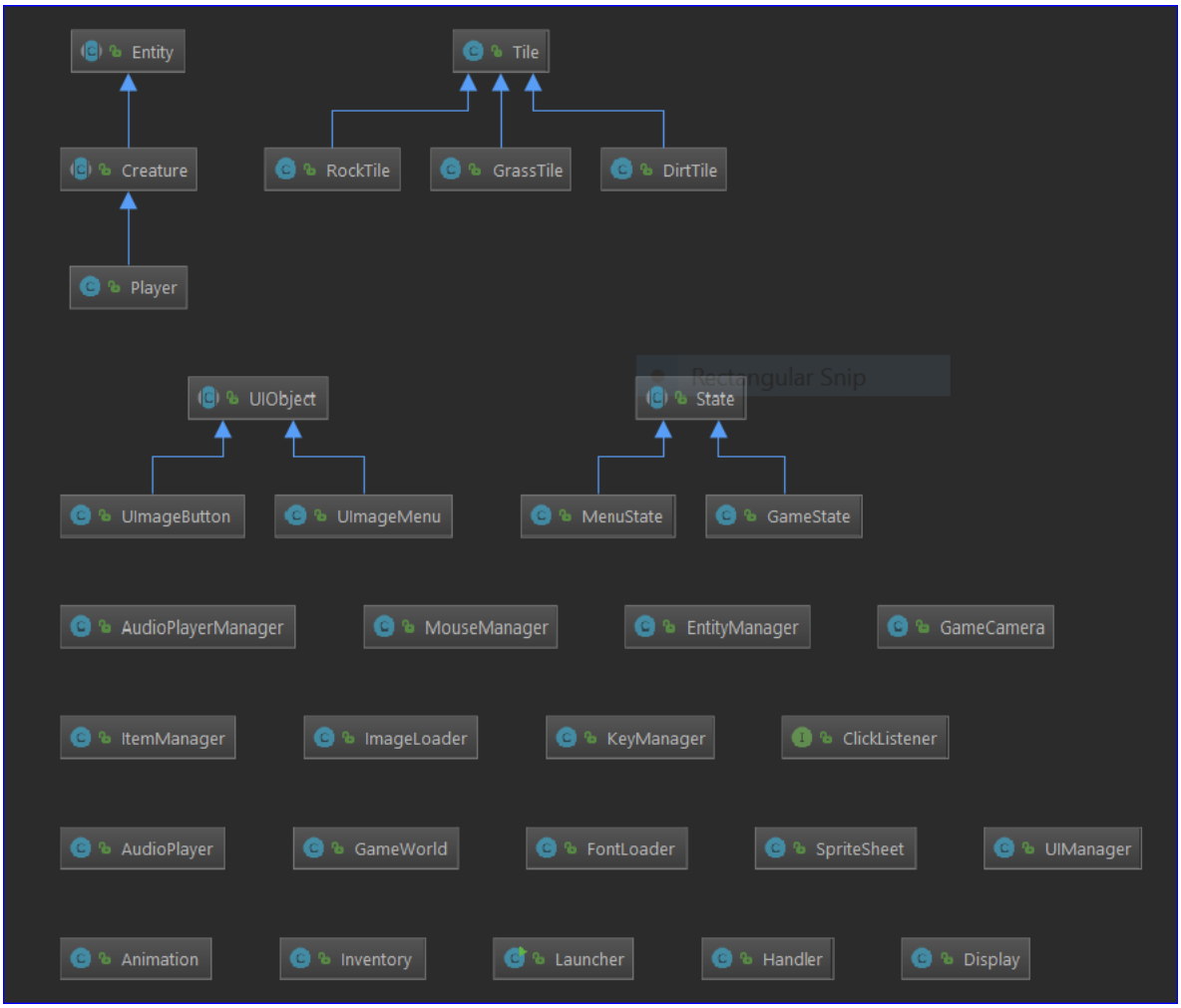
The Launcher Class is the entry point to the game.

The Game Class represents the game loop. In the game loop, all the variables are constantly updated and rendered and this process is repeated as long as the game session is running.

The Handler Class acts as a manager to manager various variable in the game.

Moreover, the details of each class are explained in section 9.

6.1 Tank Game Class Diagram



Tank Game base UML diagram only with Inheritance relationship shown.

The Player Class inherits from the Creature Class which in turn inherits from the Entity Class.

Likewise, The RockTile, DirtTile, and GrassTile class each inherits from Tile class.

Similarly, The MenuState and GameState class each inherits from the State Class.

Also, the UImageButton and the UImageMenu class each inherits from the UIObject Class.

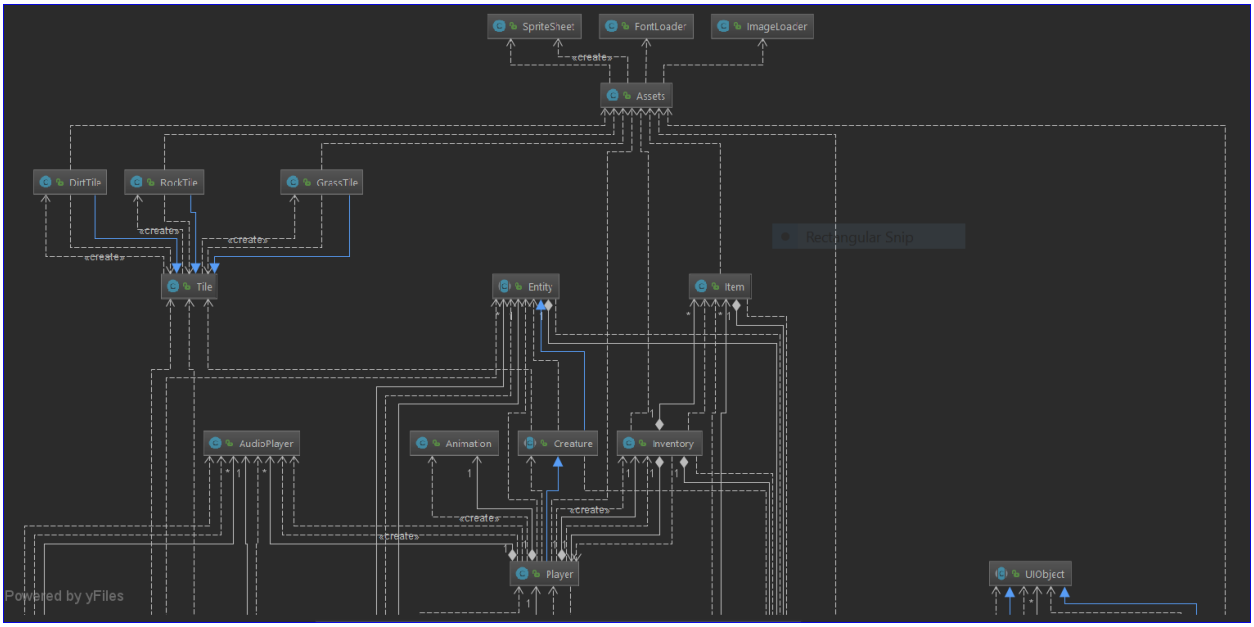
Inheritance is one of the cornerstones of OOP programming principle. Inheritance enables sub-classes to take on the properties of super class. In our first example: the Entity Class is the Super class to the Creature class and the Creature class is the super class to the player class. The Player class acts as a sub-class in relation to the Creature class and it in turn acts as sub-class to the Entity Class.

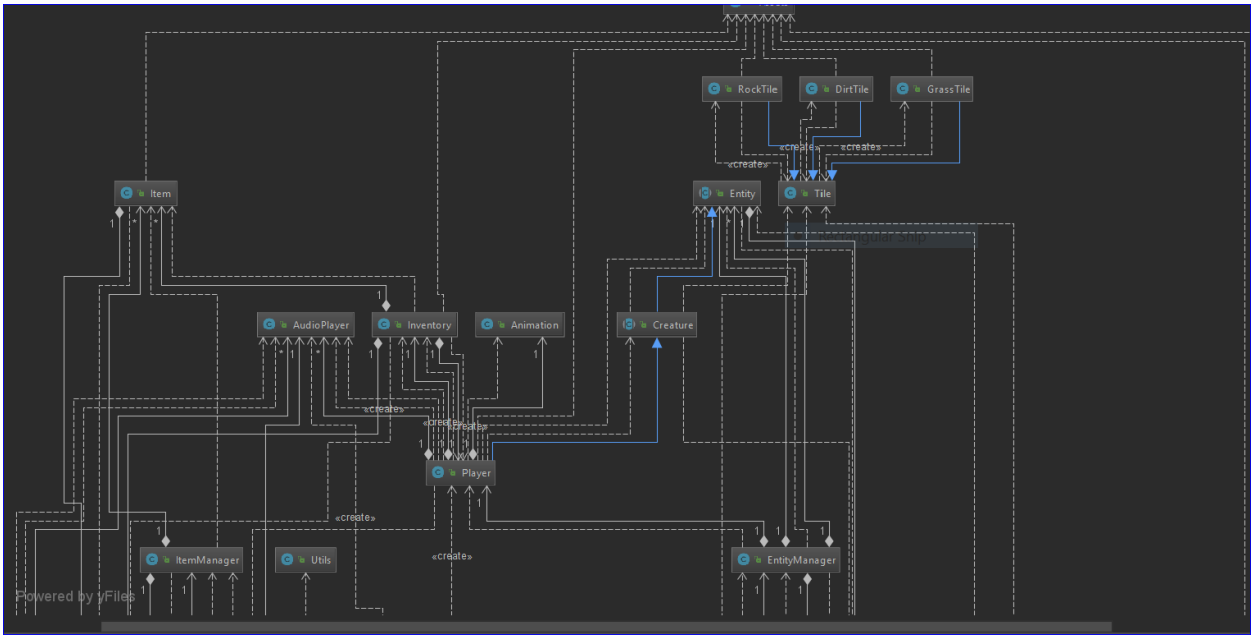
The relationship between the super class and the sub-class can be understood in terms of "IS-A" relationship. A sub-class is a specific instance of a super class. In our first example: The Player class is a specific instance of the Creature Class.

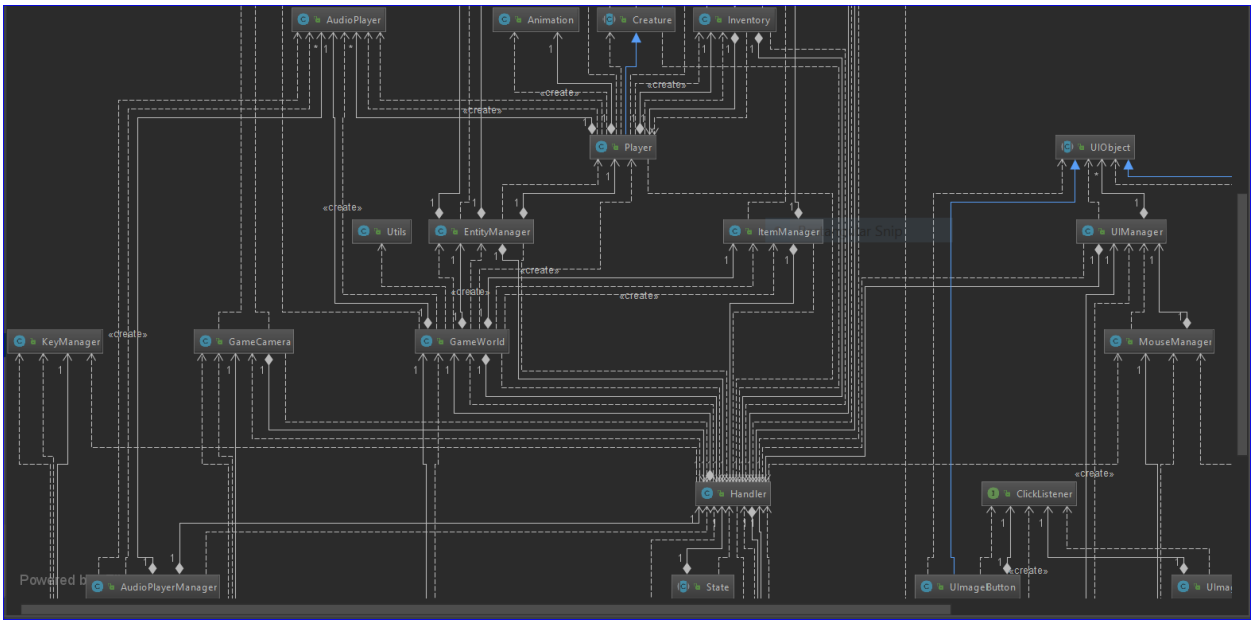
The advantages of inheritance include:

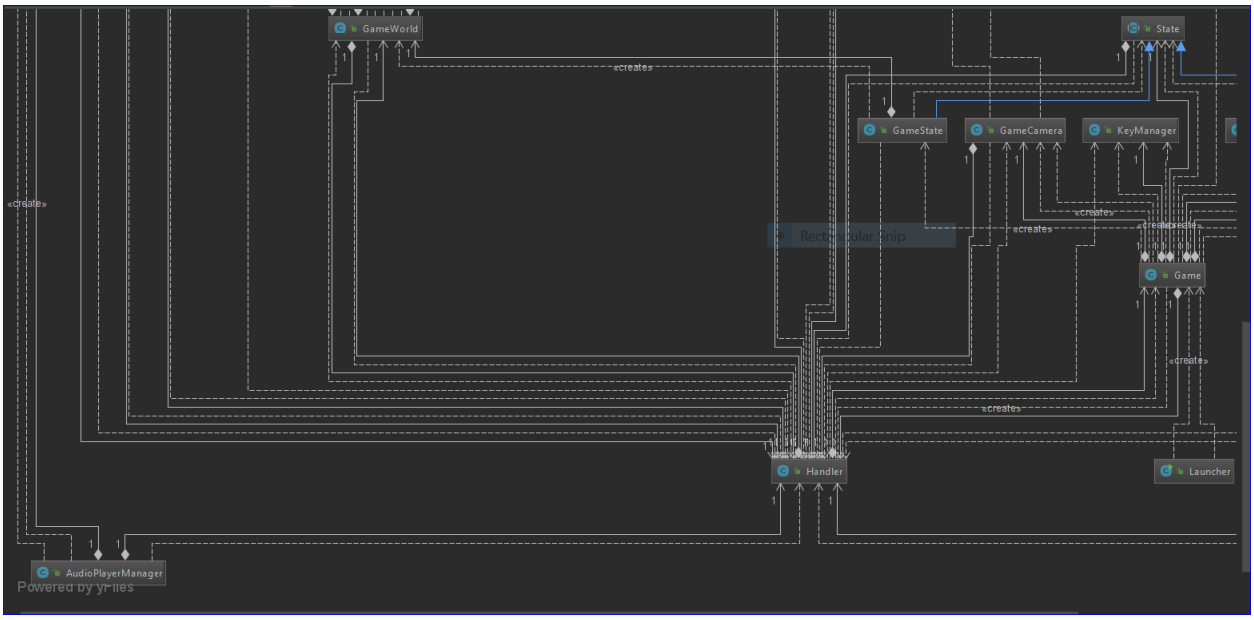
1. Defining relationships between classes,
2. Organizing classes into groups, and
3. Overriding inherited methods to implement class specific functionality.

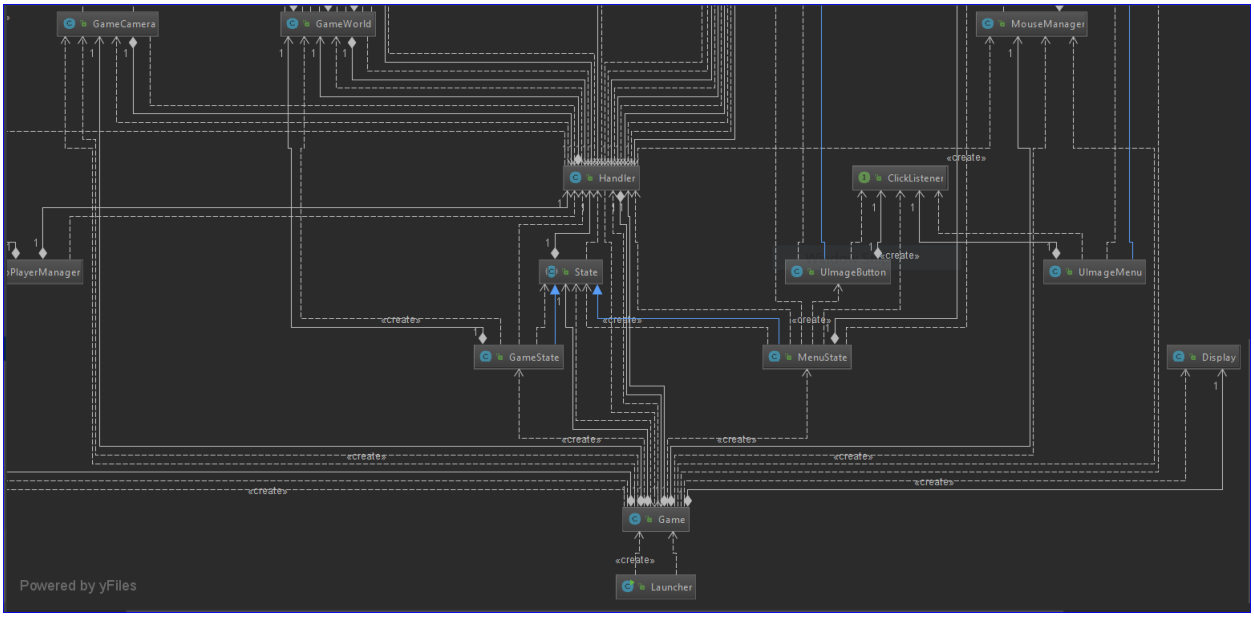
Good use of inheritance is crucial to developing effective object-oriented solutions.



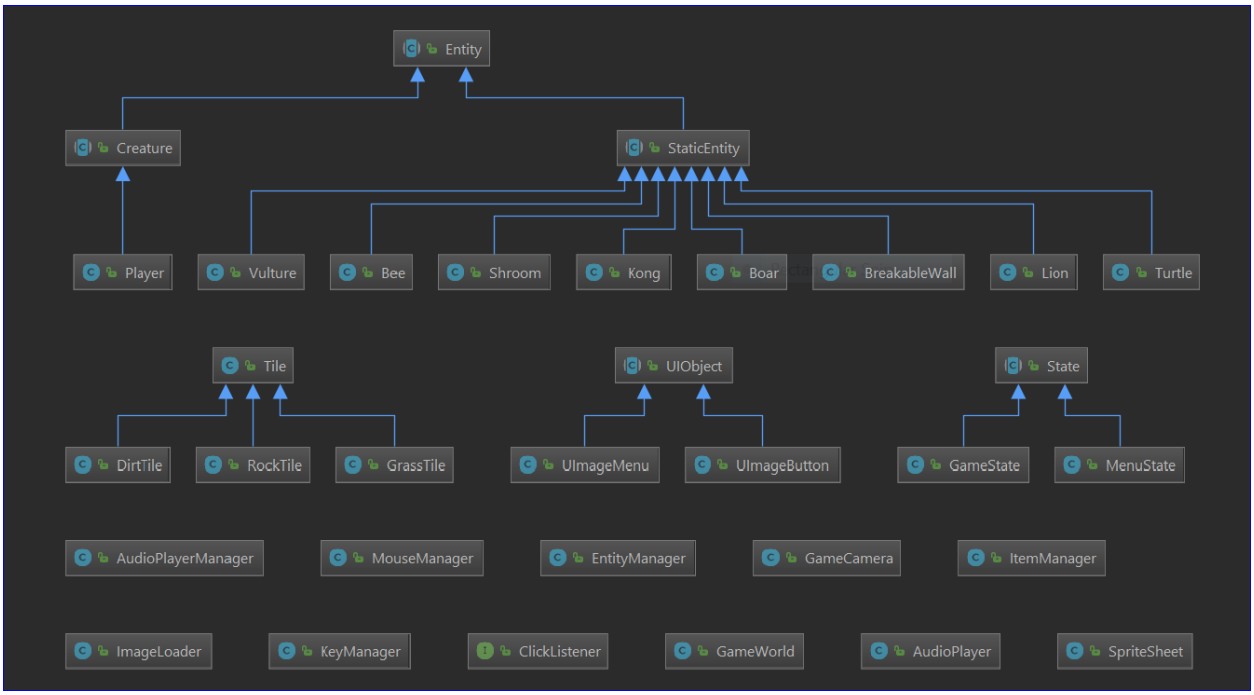

RockTile
DirtTile
GrassTile
Item
Entity
Tile
AudioPlayer
Inventory
Animation
Creature
Player
ItemManager
Utils
EntityManager
«create»
«creates»
Powered by yFiles

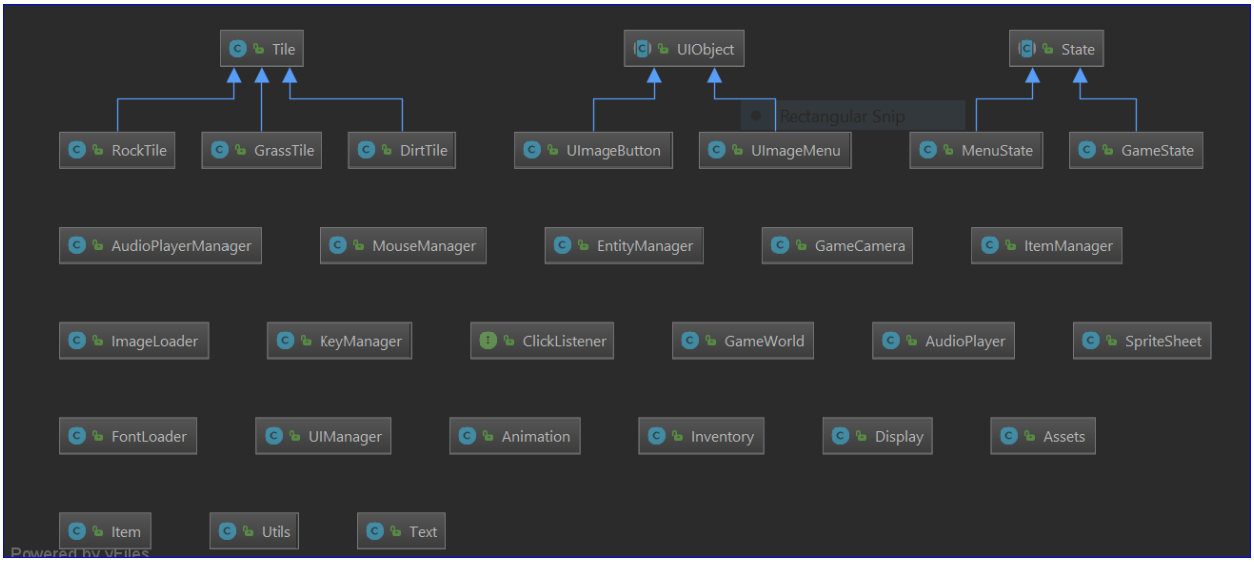

AudioPlayer
Animation
Creature
Inventory
Player
UIObject
Utils
EntityManager
ItemManager
UIManager
KeyManager
GameCamera
GameWorld
MouseManager
Handler
ClickListener
AudioPlayerManager
State
UImageButton
«create»
«creates»

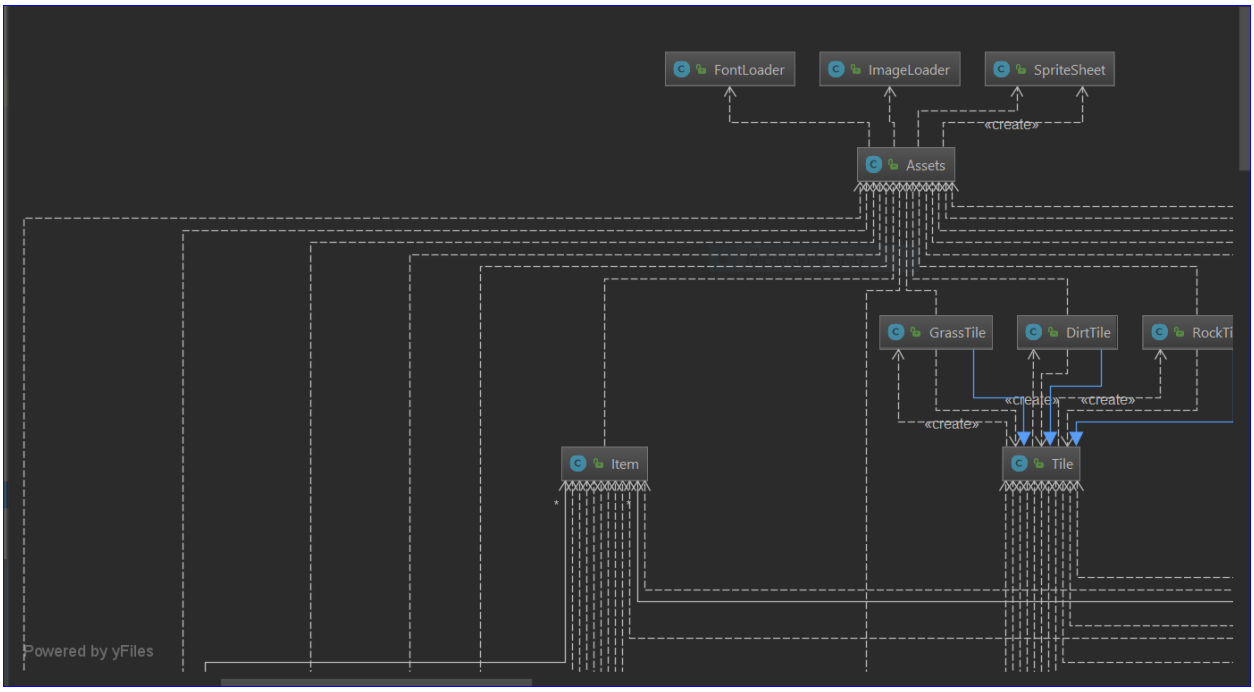

GameWorld
State
«create»
GameState
GameCamera
KeyManager
Game
Handler
Launcher
AudioPlayerManager
Powered by yFiles

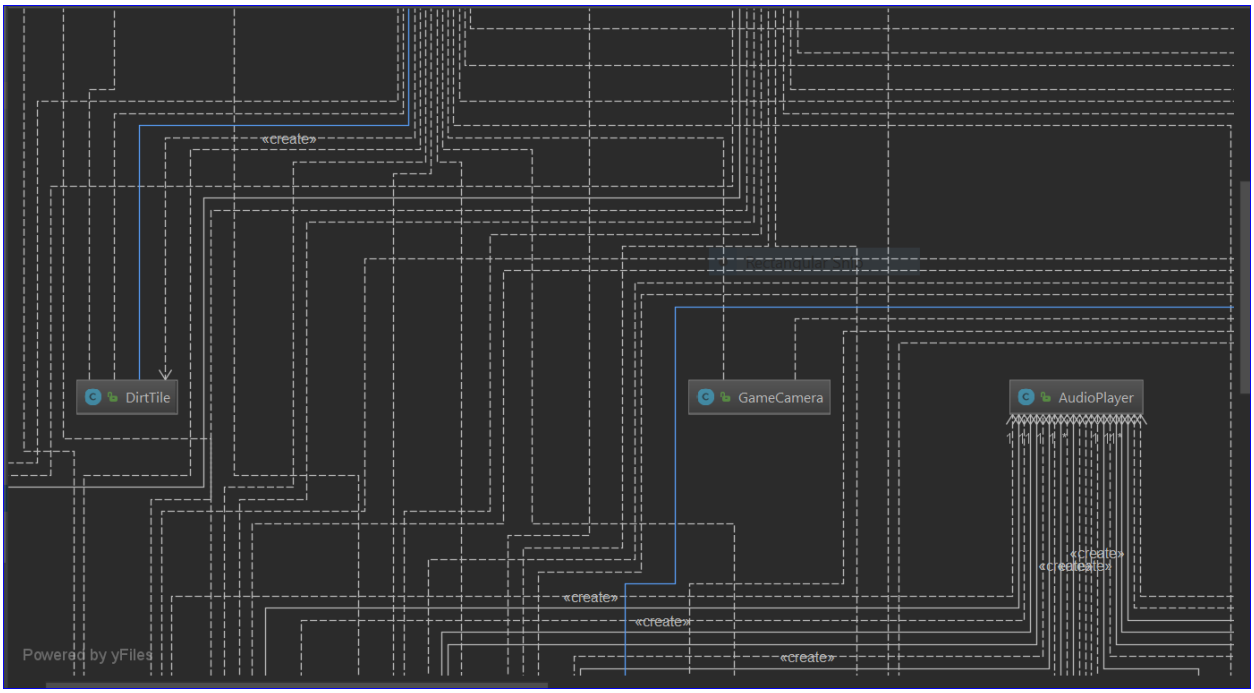

GameCamera
GameWorld
MouseManager
Handler
ClickListener
State
UImageButton
UImageMenu
GameState
MenuState
Display
Game
Launcher
«create»
Powered by yFiles

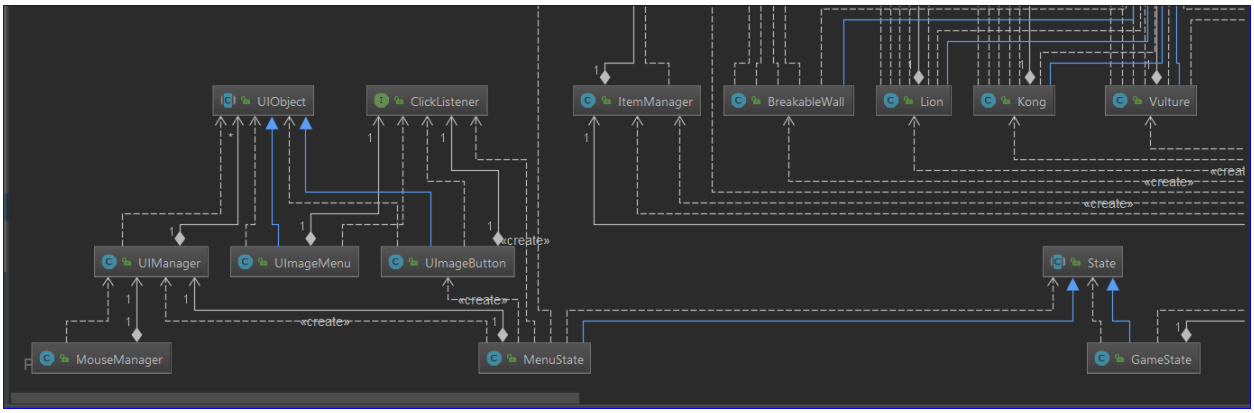
6.2 In the Den Game Class Diagram

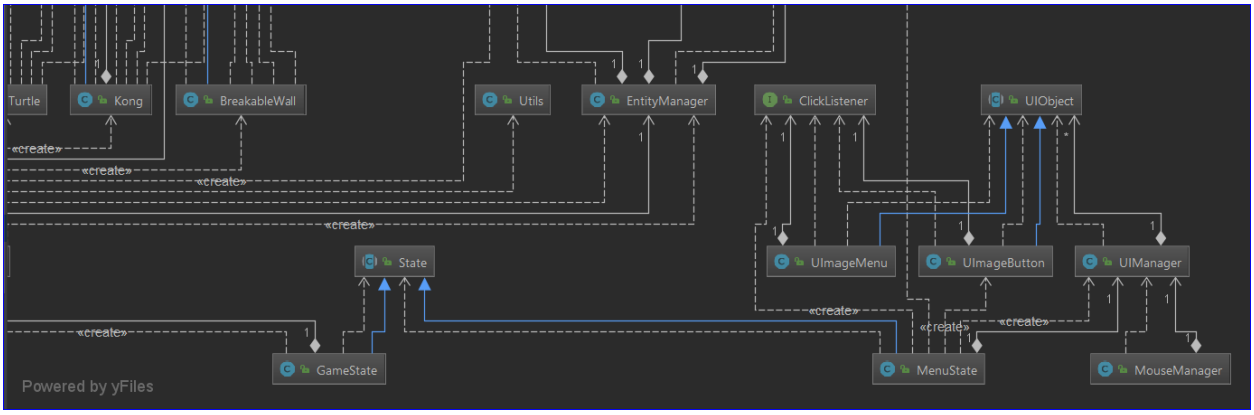


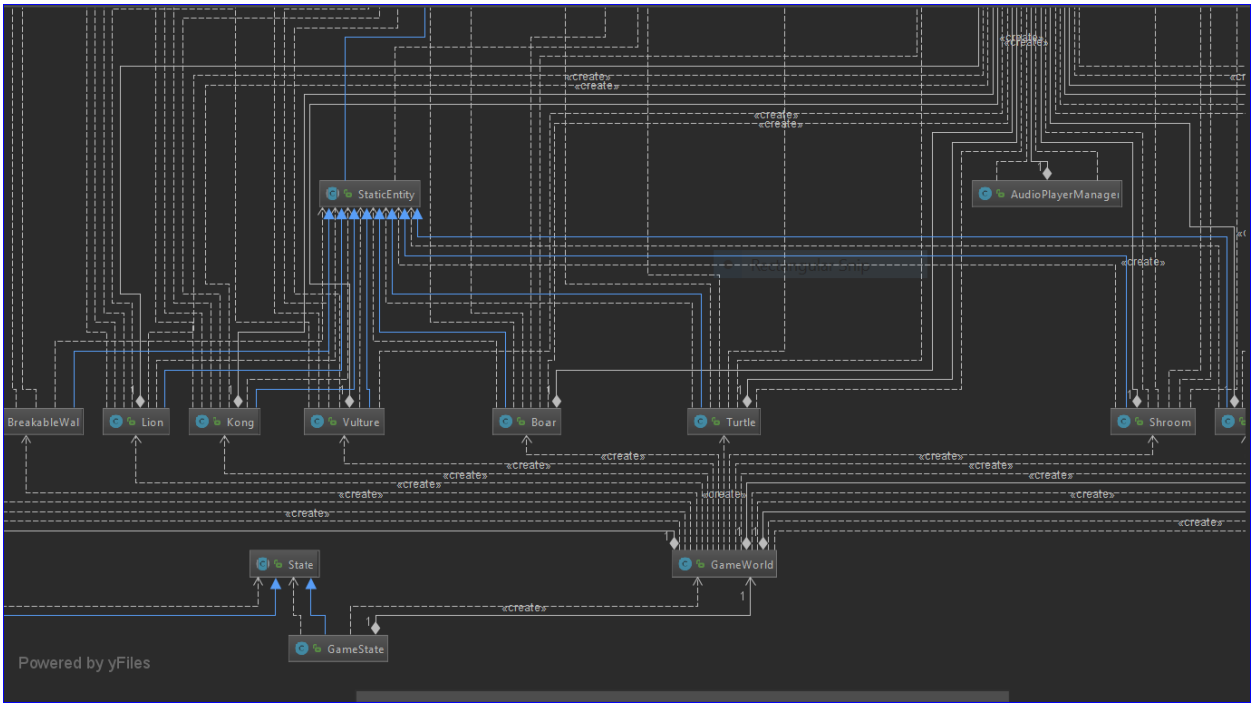



FontLoader
ImageLoader
SpriteSheet
«create»
Assets
GrassTile
DirtTile
RockTi
«create»
«create»
«create»
Item
Tile
Powered by yFiles


«create»
DirtTile
GameCamera
AudioPlayer
«create»
«create»
«create»
«create»
Powered by yFiles


UIObject
ClickListener
ItemManager
BreakableWall
Lion
Kong
Vulture
UIManager
UImageMenu
UImageButton
State
MouseManager
MenuState
GameState
«create»


Turtle
Kong
BreakableWall
Utils
EntityManager
ClickListener
UIObject
State
UImageMenu
UImageButton
UIManager
GameState
MenuState
MouseManager
«create»
Powered by yFiles


StaticEntity
AudioPlayerManager
BreakableWal
Lion
Kong
Vulture
Boar
Turtle
Shroom
State
GameWorld
GameState
«create»
Powered by yFiles

7 Game Controls

7.1 Tank Game Controls

1. Up, Down, Left and Right arrow key to move a player
2. **W + Up** arrow key to attack when a player is in up direction
3. **S + Down** arrow key to attack when a player is in down direction
4. **A + Left** arrow key to attack when a player is in left direction
5. **D + Right** arrow key to attack when a player is in right direction

7.2 In the Den Game Controls

Same controls as the Tank Game.

8 Description of Shared Classes Between Two Games

The Tank Game and In the Den Game shared a lot classes between them. In fact, In the Den Game can be thought of as an extension of a Tank Game albeit with similar structure yet with different outcomes. I focused early on to reusability of source code and accordingly I was able to achieve high reusability of source code in implementing the second game In the Den.

9 Description of Tank Game Specific Classes

9.1 Launcher Class

The Launcher Class is responsible for launching game application. It contains the main method.

9.2 Game Class

The Game Class is responsible to run the game. This class implements Runnable interface so when the game in launched it is executed on its own thread.

This class implements run() method for game performance efficiency.

9.3 Handler Class

The Handler Class is responsible to handler various variables in the game.

9.4 Display Class

The Display Class is responsible to display game window. This class implements a single createDisplay() method. The createDisplay() method initializes a canvas for drawing graphics.

9.5 Entity Class

The Entity Class is an abstract class. It is responsible to represent the various entities of the game such as player, tiles and enemies. Any sub-class extending from this class must implement three methods:

1. tick() method – updates entity co-ordinates
2. render() method – draws entity to the screen
3. expired() method – checks if the entity is alive or expired

This class is special in the sense that it also has the collision detection for the entities with checkEntityCollision() method. In this method I computed the intersection of bounding box of the specified entity and checked with the bounding box of the other entity of interest using intersects() method from Java Rectangle Class.

9.6 Entity Manager

The Entity Manger Class is responsible to manage various entities of the game world. I implemented ArrayList to hold entities objects.

9.7 Bullet Class

The Bullet Class is responsible to represent a collide-able entity bullet.

9.8 BulletController Class

The BulletController class is responsible to manage bullet objects. I implemented LinkedList to hold my bullet objects. The idea behind have the manager class to manage the bullet class was to give each class a single responsibility.

9.9 Creature Class

The Creature Class represents the various creatures with mobility in the game. This class is an abstract class and extends another abstract Entity class.

9.10 Player Class

The Player Class represents various players in the game. In the Tank Game, each tank is an instance of the player class. In the second game In the Den, the hunter is also the instance of the player class.

9.11 StaticEntity Class

The StaticEntity Class is an abstract class that extends the Entity class. It represents various static entities such as a rocket in the Tank Game and boar, bee, ape, mushroom in the second game In the Den.

9.12 BreakableWall Class

The BreakableWall Class represents a breakable wall – a static entity. It also extends StaticEntity Class.

9.13 Rocket Class

The Rocket Class inherits from the StaticEntity Class. It represents a static entity.

9.14 GameWorld Class

The GameWorld Class represents the game world. Both the static and non-static entities come alive due to this class.

9.15 Assets Class

The Assets Class is responsible to load game assets such as sprite sheet, font, and tiles. This class is significant because I combined many smaller images in each sheet and cropped them to render in the game world. This way I was able to reduce the memory usage, speed up the start time of the game. Most of all, the game performance was smooth and without any lag despite, many high-resolution images and audio files.

9.16 Animation Class

The Animation Class facilitates entity animation.

9.17 FontLoader Class

The FontLoader is wrapper class to Font class. It is responsible to load font(s).

9.18 GameCamera Class

The GameCamera Class is responsible to center entities to the center in the game world so the viewer can have a focused view of the visible window.

9.19 ImageLoader Class

The ImageLoader Class acts as a wrapper to BufferedImage Class to load the graphics.

9.20 SpriteSheet Class

The SpriteSheet Class represents a sprite sheet. This class also acts as a wrapper to BufferedImage Class.

9.21 Text Class

The Text Class is responsible to draw fonts to the screen. This class also acts as a wrapper to Font Class.

9.22 KeyManager Class

The KeyManager Class is responsible to manage user inputs from the keyboard. This class implements KeyListener interface. I implemented keyPressed() method and keyReleased() method in this class.

9.23 MouseManager Class

The MouseManger Class is responsible to manage the mouse inputs. This class implements MouseListener and MouseMotionListener interface. I further implemented mousePressed(), mouseReleased(), and mouseMoved() methods.

9.24 Inventory Class

The Inventory Class is responsible to represent inventory items in the game. Moreover, this class holds item objects in an array list. The goal is to assist player in the game with power up feature. For example, a player could have a shell or rocket in the inventory and based on the count of those items the intention was to give a player dynamic power up features.

9.25 Item Class

The Item Class represents various reward items that a player can collect in the game world.

9.26 ItemManager Class

The ItemManager Class is responsible to manage item objects. This class holds items objects in an array list. Thus, it avails features such as add and remove items from the list which other classes could take advantage of.

9.27 State Class

The State Class is an abstract class. It represents various game states. The sub-classes that extend this class must implement tick() and render() methods.

9.28 MenuState Class

The MenuState Class inherits from the State Class. This class represents the menu state of the game.

9.29 GameState Class

The GameState Class is responsible to represent the game state.

9.30 Tile Class

The Tile Class represents a tile in the game world. One of the significant method in this class is isSolid() method which checks if the tile is solid or not. If the tile is solid then the player can't step over it. By default, the isSolid() method returns false, meaning that the tile by default can be stepped over by the player. However, the sub-class overrides this method.

9.31 RockTile Class

The RockTile Class inherits from the Tile Class. This class overrides isSolid() method from the Tile Super Class. It represents a solid rock tile in the game world. A solid rock tile is un-breakable. Thus, a player can't step over it.

9.32 GrassTile Class

The GrassTiel Class inherits from the Tile Class. This class represents the grass tile in the game world. Also, the grass tile is not solid. Thus, the player can step over it.

9.33 DirtTile Class

The DirtTile Class inherits from the Tile Class. This class represents the dirt tile in the game world.

9.34 UIObject Class

The UIObject Class is an abstract class. This class is responsible to abstract out the core user interface of the game.

9.35 UIManagerClass

The UIManager Class is responsible to manage the UIObject objects by storing them in the array list.

9.36 UImageButton Class

The UImageButton Class inherits from UIObject Class. It also represents a user interface image button.

9.37 ClickListener Class

The ClickListerner Class is an interface class that represents the user click listener. Any sub-class that implements this class must implement onClick() method.

9.38 Utils Class

The Utils Class facilitates as a utility class.

10 Description of In the Den Game Specific Classes

10.1 AudioPlayer Class

The AudioPlayer Class is responsible to represent sound effect for the game. This class also acts as a wrapper class to AudioInputStream and Clip class.

10.2 AudioPlayerManager Class

The AudioPlayerManager Class is responsible to mange sound effects in the game. I implemented HashMap to map sound effect name to the sound clip.

10.3 Bee Class

The Bee Class represents a bee – a static entity in the game world.

10.4 Boar Class

The Boar Class represents a boar – a static entity in the game world.

10.5 Kong Class

The Kong Class represents a legendary King Kong – a static entity in the game world.

10.6 Lion Class

The Lion Class represents a lion – a static entity in the game world.

10.7 Shroom Class

The Shroom Class represents a mushroom – a static entity in the game world.

10.8 Turtle Class

The Turtle Class represents a turtle – a static entity in the game world.

10.9 Vulture Class

The Vulture Class represents a vulture – a static entity in the game world.

11 Implementation Discussion

Initially, building a game from scratch seemed quite a daunting task. I made good use of all the resources provided by the instructor and also countless fragmented resources in the internet.

In order to avoid getting side track I kept myself in check with the requirements by following the basic foundation of object-oriented principle (OOP) namely:

1. Inheritance
2. Encapsulation
3. Polymorphism

Besides practicing good OOP I also focused on reusability so I could use the source code from the Tank Game in building the second game In the Den game.

Moreover, I also adhered to the SOLID design principle in designing my game architect.

First, each class were designed with a single responsibility in mind.

Second, all the classes were enforced with proper access modifier to ensure they are all open for extension but closed for modification.

Third, when the sub-class were to inherit from the super class, the subtypes were designed so they could be substituted for their base types, adhering to the Liskov's substitution principle (LSV) of the design pattern.

Fourth, in order to avoid clients forced to depend on methods that they do not use, interface segregation principle of design pattern was taken into consideration.

Finally, dependency inversion principle of the design pattern was implemented so that the high-level modules do not depend on low-level module; both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions. In our class the Entity Class is one of the many abstract class. It does not depend on sub-classes. However, it abstracts out what it means to be a player.

Generally, software should be designed in such a way that any future changes will not break the entire program. I learned that following above design pattern principle was pivotal as a guide to a better design. As a result, I was able to take the entire Tank Game source code modify where needed in building the second game In the Den in half the time.

12 Project Reflection

I had never built a game before and I was looking forward to the term project. The term project experience was both fun and partly laborious. I spend countless hours combing my source code and going over resources.

My approach to building both the games was to practice good OOP and reusability of code. I'm quite positive that I was able to reuse almost all of the code from the first Tank Game to build the second game In the Den.

I was little nervous to begin the term project because I was not exposed to Java graphics component Swing. So, I concentrated my effort in getting a good grasp in Swing for the first 3 days.

Once I learned Swing, I got into building UML diagram. First it was required and second, being a visual person, it was no brainer for me to come up with working UML diagram. My first UML diagrams were not where close to being perfect but I grasped the concept of what needs to be built first to get started with. Also, instructor provided plenty resources which were very helpful.

I must confess, being able to collaborate and ask fellow mates in slack channel definitely helped me past many of the barriers.

In the Tank Game I could not implement mini-map, independent controller for each player and my bullet flew only in x-axis.

I greatly improved on my shortcomings in my second game. I was able to add sound effects, animation, and score info that updates.

I find plenty room to improve my game designing skill. That also means I'll find ample room to grow as a better game developer as I harness my skill in the coming years.

13 Project Conclusion/Results

The term project was definitely my favorite project so far. It really stretched my learning path. The project was definitely challenging. However, I strive to learn from my mistakes and success. I strived to learn as much as I can from building two game rather than just focusing on final grade. I must confess that I have learned a lot and come a long way to become a better programmer in designing better software in these projects.

The game development process was a fun way to explore Java's hidden gem of rich language features which I absolutely enjoyed and at the same time learned a lot.

14 Credits/References

1. INST. Anthony Souza
2. Slack Channel: SFSU Computer Science
3. Game sprites: https://opengameart.org (Creative Commons CC license)
4. Sound effects: https://freesound.org (Creative Commons CC license)
5. Habgood, Jacob., et all. *The Game Maker's Apprentice: Game Development for Beginners*. Apress; 1 edition, 2006.