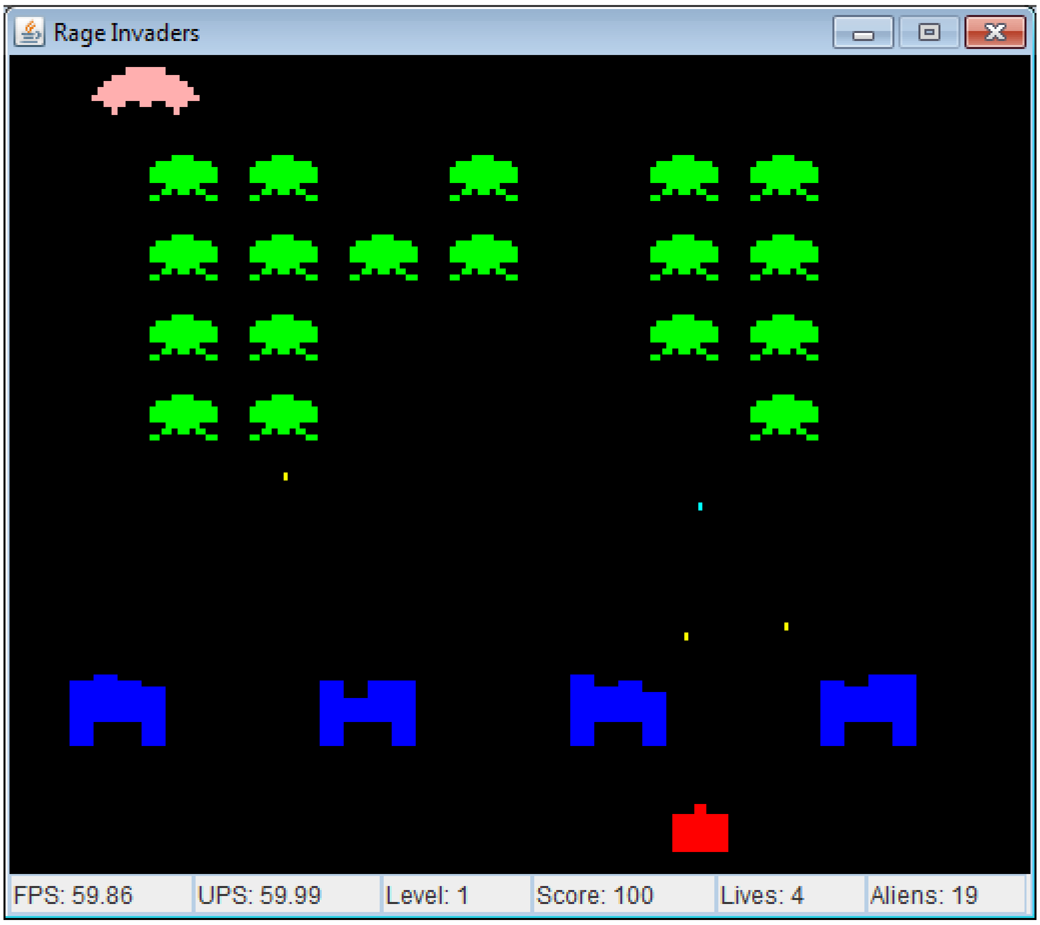
CS1006 Project 2- Rage Invaders

Introduction

This project was about creating a video game in the style of classic shooters such as Space Invaders or Galaxian. The basis of the game is that the player controls a cannon, which fires missiles at the approaching aliens. They must destroy all the aliens before they either destroy the cannon or reach the bottom of the screen. This repeats throughout all the levels until the player is defeated and then his score is recorded. The title of the project being submitted will be referenced as “Rage Invaders” throughout the rest of this report. This version of the game is notably more difficult than the original, which was intended.

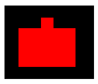
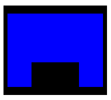
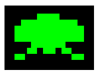
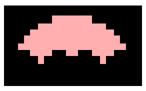
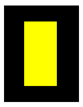
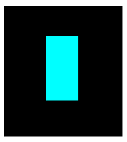
Sample Output

This sample output demonstrates some of features of the Space Invaders game:



FPS: 59.86 UPS: 59.99 Level: 1 Score: 100 Lives: 4 Aliens: 19

Explanation of sample output

Object	Explanation	Picture
Player Cannon	The object the player controls. Is used for shooting the aliens. Has a limited number of lives.	
Barricade	The barricades provide cover for the player. They slowly degrade when shot by either the player or the alien, there are four of them on screen and they regenerate every level.	
Alien	The players enemy there are 28 on screen to start with. They fire missiles at the player throughout the game.	
Special Alien	The special alien flies across the screen every 10-20 seconds. There are special bonus points given for shooting it.	
Enemy Laser Missile	The missiles the enemy fire at the player. If the player is hit they lose a life, if they lose all their lives they die and it is game over.	
Player Laser Missile	The missile the player fires at the aliens. If an alien is hit by one of these they die.	

The outline of the polygon graphics for the special ship and the alien were taken from a flash version of the game[i].

Design

When considering the design for a video game there are two things to consider; the design of the gameplay i.e. how the game plays, and the design of the actual code and how this is implemented.

Gameplay

When designing a shooter there is one major factor to consider: difficulty. The entirety of the gameplay revolves around the difficulty the creator of the game is aiming for because a large proportion of the decisions made in relation to gameplay affect the difficulty.

When considering difficulty there are many things to take into account: the rate at which the enemies fire, how far and often they move, the rate at which the player is allowed to fire etc. How these are designed will determine the difficulty of the game and the rate at which difficulty increases.

Enemy Fire

The rate at which the enemies fire, and the rate at which that is increased obviously has a direct influence on the difficulty of the game. The more missiles the enemies fire, the harder it is because it makes it harder to avoid being hit.
In Rage Invaders, the enemies fire five percent more often every time the player advances a level. This means the initial delay between fired shots will be slow but will increase as the player advances. Furthermore due to it increasing by a percentage, the rate of increase will also greaten every time the player advances further.

Enemy Movement

Since the game ends if an alien reaches the bottom of the screen, the speed of their movement has an effect on the difficulty; the faster they move, the faster the player has to kill them if he wants to survive.
In Rage Invaders there is a set interval at the start of each level between each movement of the enemies. This interval is decreased every time the player kills an enemy. Therefore as the player progresses the speed of the remaining enemies increases. Also, as the player advances a level the initial rate of decrease is increased.

Number of Enemies

The number of enemies on screen has a direct effect on the difficulty of the game. The more there are, the longer it takes for the player to clear a level and obviously the more missiles they have to avoid. In Rage Invaders the number of enemies is set to a 7x4 formation. Whilst this sounds small, later in the game this becomes more than enough to cope with.

Player’s Rate of Fire

The player’s rate of fire essentially determines how fast they can kill the enemies. Therefore the more this is limited, the harder it is for the player to progress. In Rage Invaders the player is limited to only being able to fire a shot when one of his own is not on screen.

Player's Movement

The speed at which the player moves has a direct influence of the difficulty of the game. On one hand if he is slow it makes it harder for him to dodge bullets and at times get to where he needs to be, to hit certain aliens, but at least it makes it easier to control. On the other if he is very fast it makes it a lot easier for the player to dodge and hit the enemies, but he becomes unwieldy and hard to control. Also it became quite common in later vertical shooters to allow the player to move in bow the x and y axis. This meant there were no barricades but allowed for more freedom of movement.
In Rage Invaders, the speed is set to a reasonably slow paced speed, fast enough that the player can dodge bullets, but fast enough that the controls aren't horribly unwieldy. Also since Rage Invaders is quite true to the original Space Invaders game there are is no movement in the y-axis.

Player Size

The larger the player is, the easier it is for the aliens to hit them. If the player is too small the harder it is for the aliens to hit them. This means that the size of the player is very important to the difficulty of the game. In Rage Invaders, the cannon is quite small. This was done because whilst again it makes the start of the game relatively easy, later on big cannon would be impossible to use as it wouldn't be able to dodge missiles at all.

Game Type

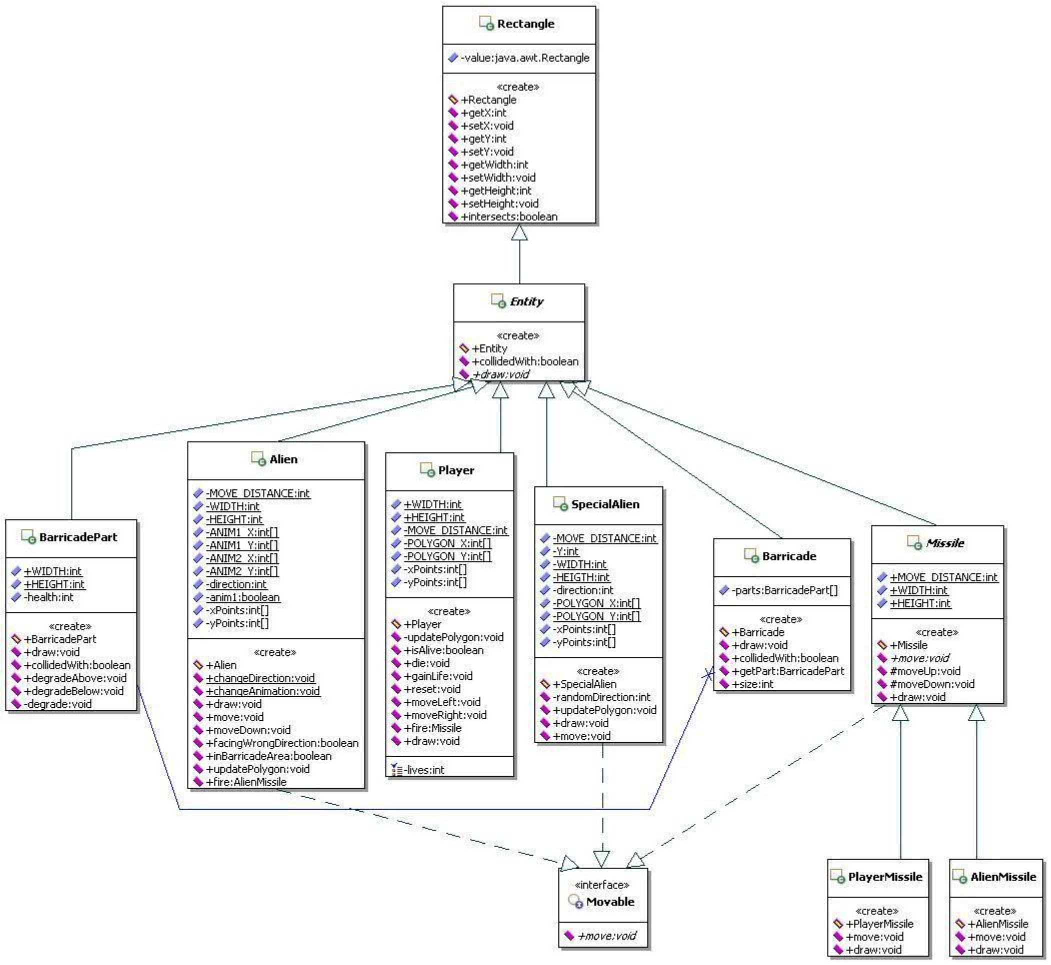
The type of game created has an effect on the difficulty of the game as it sets some parameters for how difficult the game can be made. For example the original fixed shooter[ii] games are nowhere near as difficult as the later bullet-hell (also known as manic shooter) games that came around in the nineties.
Rage Invaders is a mixture, of both fixed shooter and bullet-hell. It starts off similar to the original game but as the player progresses the game becomes steadily harder and begins to have more in common with a bullet-hell style game as the screen becomes almost full of enemy fire. The barricades also come back at the start of each level unlike the original where once damaged they stayed damaged. This was done because of the bullet-hell aspect later as it would be ridiculously unfair to play the later levels with no barricades.

Game Engine

When designing the game engine, great consideration has to be given to data representation, which classes need to be created and their purpose. Furthermore certain design decisions have to be made in deciding the implementation of certain features, as there are always several ways of doing so.

UML Diagrams

There are additional UML diagrams in the "UML"-folder, but here is the most notable one (alien.entity). It illustrates the hierarchical structure of the implementation:



Note: The Rectangle class was used as a wrapper for a java.awt.Rectangle, which has all its fields public. The reason for its use as a superclass is its usefulness in the context of collision handling, which is discussed later.

Explanation of Classes

Class	Purpose
alien.Constants	Stores most of the constants used throughout the rest of the program.
alien.GameFrame	Creates the frame used to run the game. Contains the main method which is used to run the program.
alien.GamePanel	Used for displaying and processing elements from the game and the user input.
alien.Random	Used for generation of random numbers. Used throughout the program.
alien.Sound	Used for playing the in-game sound effects.
alien.entity.Alien	Used to represent the behaviour of the aliens appearing on screen.
alien.entity.AlienMissile	Represents the alien missiles that are fired by the aliens. If a player is hit by one he loses a life and if a barricade is hit by one it degrades. Extends Missile
alien.entity.Barricade	Represents the barricades. Is made of barricade parts which degrade when shot.
alien.entity.BarricadePart	Are parts that make up an entire barricade. They degrade every time they are hit and on the fourth hit degrade completely.
alien.entity.Entity	A superclass of all visible elements on the screen. Extends rectangle.
alien.entity.Missile	Represents the weapon fired by both the aliens and the player. Is a superclass of both missile classes.
alien.entity.Movable	An interface that specifies if an object can be moved around on the screen.
alien.entity.Player	Represents the player's controls and it's representation on the screen.
alien.entity.PlayerMissile	Represents the missile fired by the player. If an alien is hit by this it dies, if a barricade is hit by this it degrades. Extends the class Missile.
alien.entity.Rectangle	Wrapper used to wrap java.awt.Rectangle. Provides basic getters, setters and a method of collision detection. Is the superclass of the Entity class.
alien.entity.SpecialAlien	Represents the special alien that flies across the top of the screen. Determines what direction it flies in from.
alien.hiscore.Hiscore	Class for management of the high scores, generates the table, loads it and checks if a score is eligible.
alien.hiscore.Hiscore	Creates a score object containing the players name and their score
alien.exception.ScoreFormatException	Generated if the high scores are not formatted correctly.

Data Structures

Aliens

In GamePanel, Aliens are contained in a one-dimensional array. While it may seem more logical to represent them in a two-dimensional array, there is one important benefit of dealing with then in this manner. Having them in a one-dimensional array makes it easier to iterate through the array for movement and collision detection, while still enabling the possibility of treating the aliens as rows and column.

Sound

The sound files are storied in byte arrays and a ByteArrayInputStream is created each time a sound is played. This implementation was inspired by email correspondence with Ian Miguel[iii]. The original basis of the sound implementation was based on a information from a tutorial on using the sun.audio package[iv] and the sound files were found on djgallagher.com[v]. Whilst it may have been more efficient to use InputStream.mark and reset methods, there was difficulty in getting them to function as intended.

Collision handling

Collision handling is done using the java.awt.Rectangle.intersects method. Having all Entities as subclasses of alien.entity.Entity (a subclass of alien.entity, which is a wrapper for a java.awt.Rectangle) makes it possible to do easy collision handling:

```
public boolean collidedWith(Entity other) {
      return intersects(other);
}
```

If two rectangles intersect, they are bound to have collided, and the appropriate action should take place. I.e. If an alien has collided with a missile, it should die and disappear from the screen.
A better way for this to be done would have been using the java.awt.Polygon.intersects method. This would mean that the Player, SpecialAlien and Alien classes would have to inherit certain operations from java.awt.Polygon. It could have been implemented using another wrapper.

Enum Game States

The states of the game are set using an enumeration type. There are six possible states. The table below explain when they occur and what they mean

State	Meaning	When
WELCOME_SCREEN	The welcome screen of the game is displayed	When the game is opened.
NEW_GAME	A new game has been created, but not started	After the player presses space when the game is on the welcome screen.
PLAYING	The user is playing the game	After the player presses space when it is in the NEW_GAME state. After un-pausing.
LEVEL_PAUSE	The pause after each level: “Level clear. Press SPACE to continue.”	After a level is completed successfully, i.e. all the aliens are killed.
GAME_OVER	The game is over	Happens if the player loses all its lives or the aliens reach the barricade area
PAUSED	The game has been paused	When the window loses focus or one of the pause game buttons are pressed.

The use of enumeration type to store the state makes it easier to control what happens in each different state, as only this one variable has to be checked, rather than having to check a number of different variables.

Testing & Limitations

During testing several errors or “glitches” became apparent. Firstly when playing if you pause the game it resets the timer on the shots. This was implemented to avoid the player pausing and then resuming the game to be hit with a barrage of shots. This can be abused by the player because they can manually reset the counter during gameplay. Hence, if the player had an auto-keyboard macro[vi], a program that repeatedly presses a key at set intervals, that pressed the “p” key twice every fifth of a second then no shots would be ever be fired by the aliens and the player could gain a very high score.

Similarly there is a problem with the special alien. Every time the player pauses, the timer for when the special alien should appear resets. This is because since the timer is currently set to continue running whilst paused, without the reset a player could keep pausing and waiting the right amount of time to force a special alien to appear and essentially gain a very high score without much skill.

Also, when the player is down to one alien they can add to their score counter by waiting for the alien to appear and again gaining lots of points without much skill.

There is also a limit on the number of characters that a user can type in for his name. This was done to stop the painting of the username to run over the painting of the score.

Since the hi-scores are saved in a file in the same folder as the game is and this file is not encrypted in any way, it is easy for the user to edit them. This could have been made more difficult by storing the hi-scores in the user directory or, alternatively, online. The former could have been done using the System.getProperty("user.dir") method, and the latter by using either a CGI-POST method or another type of centralized server.

Additionally, when entering hiscores, there is a limit on the number of characters that a user can type in for their name. This was done to stop the painting of the username to run over the painting of the score.

Conclusion

In this project we created our own version of the classic arcade game Space Invaders, titled "Rage Invaders". The original game involved the player controlling cannon and shooting the approaching alien invaders. Ours stays pretty true to the original with a few minor adjustments, such as the difficulty being increased. The original game did not have the barricades regenerating after every level; we decided to have them reload as otherwise the difficulty after a couple of levels would be incredibly unfair. There was also the possibility to add to the game with additional features such as power-ups but since our implementation was designed to be fairly true to the original, features like this were left out.

Future Improvements

Addressing the Limitations

If more time were available the issue with the timers could be resolved by implementing a pause on the counters, this would allow the game to be paused and would eliminate a lot of exploits dishonest players could use.

Furthermore, a method stopping a special alien from being generated could be implemented if there was only one alien left. This would stop players waiting at the end and increasing their score by taking easy shots at the special alien. Other ways this could be done would be to increase the spawn timer (i.e. the time it takes for it to appear) each time the special alien disappears, and then reset it at the end of the level, or having the special alien only appear once per level.

Online Scoreboard

We tried to use java to work together with CGI and use the computers in the lab to store and handle the high-scores. We managed to get the server-side aspects working as intended, but could not figure out how to get java to send the information using the post method with the limited amount of time to complete the project. This would be something we would be interested in implementing in the future.

Hi-score name length

In the current implementation the name entered onto hi-scores is limited to seven characters. This could have been changed by painting the scores further along the screen so that longer names could be written.

Sound

Currently the game uses .au sound files for playing the sounds. If more time was available we would look into having the game play higher quality sound files and also store the files in memory so that it doesn't have to read the file every time it plays it.
Also we'd advance on the sound effects, adding in special sound effects for certain achievements i.e. five kills in five seconds could play a "killing spree" sound in the style of Unreal Tournament

Additional Features

Features like power-ups were suggested but since our aim was to stick to the original, aside from a few adjustments, some of these were left out. In the future we might possibly make new versions of the game or sequels that would have these abilities such as control in more than one axis and power-ups, such as weapon upgrades and extra lives.

References

[i] Neave, Paul. "Free Space Invaders". Free Video Games. 08/03/10
<http://www.freespaceinvaders.org>.
[ii]Game Genres:Shmups Lecture by Professor Jim Whitehead,22/03/10
<http://www.soe.ucsc.edu/classes/cmps080k/Winter07/lectures/shmups.pdf>
[iii] Email correspondence in Appendix Folder
[iv] Chong Ser Wah and John D. Mitchell. "Java Tip 24: How to play audio in applications". InfoWorld.
08/03/2010 <http://www.javaworld.com/javaworld/javatips/jw-javatip24.html>.
[v] "Classic Arcade Games- Space Invaders". 08/03/10
<http://www.djgallagher.com/games/classics/spaceinvaders/sounds.php>.
[vi] 23/03/2010 <http://www.auto-keyboard.swiftpaste.com/>.