CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The first version of the arcade game Space Invaders was released in 1978 and was designed by Tomohiro Nishikado. Taito was the company that manufactured and sold the product within Japan. Bally's Midway division was licenced to distribute the product internationally. The first game of its kind, Space Invaders, was a fixed-shooter that established the standard for the shoot 'em up genre. The objective of the game is to rack up as many points as possible by using a laser that moves horizontally to dispose of wave after wave of falling aliens.

The video games Breakout (1976) and Gun Fight (1975) designed for the North American market, in addition to works of science fiction such as the novel The War of the Worlds (1897), the anime Space Battleship Yamato (1974), and the film Star Wars (1977), served as sources of creativity for the game's designer, Nishikado (1977). In order to finish the game's development, he needed to construct specialised hardware as well as tools for developers. After its initial release, Space Invaders was an immediate commercial success. By the year 1982, the game had grossed $3.8 billion, and it had made a net profit of $450 million. As a result, it became the entertainment product with the largest revenue at the time, as well as the video game with the highest revenue of all time. It was also the best-selling video game of all time.

It is widely acknowledged that Space Invaders was one of the most influential video games ever created. It was the beginning of a prosperous era for video games played in arcades. It has served as a point of departure for a great number of video games and game designers working in a variety of genres, and it has been ported to and distributed in a number of different forms.

1.2 HISTORY

Chat apps actually date back to the 1980s, when CompuServe released CB Simulator, the first dedicated online chat service that was widely available to the public. The first SMS was launched by Vodafone GSM in 1992, then throughout the mid to late 90s, messenger and chat rooms such as MSN and Yahoo! became popular. The mid-2000s saw us with desktop-based apps such as Skype and Google Talk, allowing users to talk face-to-face with people across the world. Chat apps as we know them today are a response to our desire for mobile communication abilities. Those old chat rooms were built for people sitting at a desktop or laptop, not with our phones or tablets in mind. As internet and data speeds have vastly improved, the experience of communicating by mobile has grown as well, allowing better features and capabilities. Part of the push toward mobile chat apps came because the mobile phone networks were still charging for every SMS sent. Using data to operate a chat app became a more appealing option for many users. By 2014, WhatsApp, which remains one of the most popular chat apps, had increased its feature set to include sending pictures and a host of other things and had grown its user base to 650 million active users worldwide in a period of five years. Facebook Messenger hit 1 billion users by the middle of 2016. Over this period, we also saw the growth of a different market that was not consumer-focused – that of enterprise chat. Early contenders in this space included Yammer and Clearspace, but it wasn't until the 2010 launch of HipChat that traction started to be gained in the enterprise marketplace. In August of 2013, a young company developing an online game decided to pivot away from this venture, and productize the internal tool they had created for communication – Slack was born. Since then, Slack has reached a $5 billion valuation, and other companies are turning out their own enterprise chat apps in competition, such as Cisco Spark, Google's new Hangouts Chat, and MS Teams. In China, WeChat has an enormous following, a big player in a market where chat apps have been even more popular than in the US. Today's chat apps are vying for the conversation between businesses and clients, between friends and family, and between colleagues at work.

1.3RULES

Basic gameplay: The player controls the cannon at the bottom of the screen, which can move only horizontally. The aliens moves both horizontally and vertically (approaching the cannon). The cannon can be controlled to shoot laser to destroy the aliens, while the aliens will randomly shoot towards the cannon. If an alien is shot by the cannon, it is destroyed; if the cannon is shot, one life is lost. However, the position of the aliens will not be reset if the cannon is lost. The initial number of lives is three.

Alien behavior: The aliens are aligned in a rectangular formation, floating slowly in horizontal direction. As the formation reaches one side, the aliens approach the bottom by a certain amount and start floating in the opposite horizontal direction. The aliens moves faster and faster as they come closer to the bottom. Any column of the alien may shoot a laser towards the cannon at a random time.

Scores: Each eliminated alien is worth 10 pts.

Completing a level: When all aliens are eliminated, the level is completed and a congratulation screen is displayed within the playfield. If the player's score is higher than the stored high score, the new high score is stored. No signature (i.e. record holder name) is needed for the high score. After the player hits any key, the game is reset to the title screen.

Game over: When all lives have been lost, or the aliens have reached a certain vertical position (successfully invaded the planet), the game ends and a game over screen is shown in the playfield. If the player's score is higher than the stored high score, the new high score is stored. After the player hits any key, the game is reset to the title screen.

1.4 Objective of the game

The objective of the game is simply to destroy a formation of advancing enemy spaceships. The player moves on to the next level upon destroying all enemy ships. Each consecutive level gets tougher and tougher with enemies shooting at the player more frequently. The number of points obtained per enemy destroyed increases with every level. The player should aim to maximize his or her score.

1.5 BENEFITS OF USING JAVA

- Java is Secure. Java achieves the protection by confining a java program to the java execution environment and not allowing it to places extraordinary demands on a page, because the programs must execute reliably in a variety of system. The hard-to-track-down bugs are simply impossible to create in Java.
- Java is multithreaded. Java was designed to meet the real-world requirement of creating interactive, networked pages. To accomplish this, Java supports multithreading programming, which allows you to write pages that perform multiple tasks simultaneously.
- Java is dynamic. Java programs carry with them substantial amount of run-time type information that is used to verify and resolve accesses to objects at run time. This makes is possible to dynamically link code in a safe and expedient manner.
- Java's exception handling. Java's exception handling avoids the problem of checking errors and handling them manually and brings run time error management into object-oriented world. Access to other parts of the computer. Java programs can be dynamically downloaded to all the various types of platform. Thus, it is portable.
- **Java is robust.** It is a strictly typed language; it checks your code at compile time. However, it also checks your code at run time. Thus the ability to create robust program was given high priority in the design of java. The multi-platform environment of the web

CHAPTER 2
LITERATURE STUDY

1. IRJET-SPACE INVADERS: AN EDUCATIONAL GAME IN VIRTUAL REALITY

Author

The literature survey conducted by the people of Department of Computer Science and Engineering, BNM Institute of Technology, Karnataka, India

Inference

This project describes the creation of a web application that uses a space environment design to implement a virtual reality game, using a VR headset consisting of goggles to create a virtual screen. The developed game allows for a greater degree of user control, thereby increasing user convenience. It enables them to visualise the abstract environment clearly, thereby improving performance. The success of this game was contingent upon the implementation of a number of concepts on an a-frames platform. The developed game allows for a greater degree of user control, thereby increasing user convenience. It enables them to visualize the abstract environment clearly, thereby improving performance.

2. AI-assisted game debugging with Cicero

Author

The literature survey conducted by the people of New York University

1. Tiago Machado
2. Daniel Gopstein
3. Andy Nealen

Inference

We present Cicero, a mixed-initiative application for prototyping two-dimensional sprite-based games, including shooters, puzzles, and action games. Cicero offers a multitude of features that can be of assistance in a variety of situations. The phases of the game development procedure. Notable features include AI agents for simulation of gameplay, a game mechanics recommender system, a playtrace

aggregator, heatmap-based game analysis, a sequential replay mechanism, and a query system that allows searching for specific interaction patterns.

3. The Mood Game - How to Use the Player's Affective State in a Shoot'em up voiding Frustration and Boredom

Author

The literature survey conducted by the people of Media Informatics of Group University of Regensburg in Regensburg, Germany

1.David Halbhuber

2. Jakob Fehle

3. Alexander Kalus

Inference

In this demonstration paper, we offer a shoot-em-up game similar to Space Invaders dubbed "Mood Game" that includes players' affective state into the game's mechanics in order to enhance the gaming experience.

Improve the gaming experience and avoid undesirable feelings such as frustration and boredom. We've built a gaming logic that adjusts the playing difficulty based on the player's emotional state by monitoring facial expressions, self-evaluation, keystrokes, and performance measurements. The developed algorithm dynamically adjusts the enemy spawn rate and enemy behaviour, the number of obstacles, the number and kind of power-ups, and the game speed to ensure a fluid gaming experience for players with varying skill levels. Future research will evaluate the efficacy of our dynamic game balancing technique.

CHAPTER 3

EXISTING SYSTEM

When Nishikado finished the game, it was met with a mixed reaction from Taito and amusement arcade owners. His coworkers praised it, applauding his achievement while waiting in line to play, whereas his bosses predicted low sales because games ended faster than other timer-based arcade games at the time. Many amusement arcade owners initially rejected the game, but it quickly caught on, with many parlours and alleys clearing space for more Space Invaders cabinets. Space Invaders became popular in Japan in the months following its release, and specialty video arcades opened with only Space Invaders cabinets.

Taito had installed over 100,000 machines and earned $670 million in Japan alone by the end of 1978. By June 1979, Taito had produced approximately 200,000-300,000 Space Invaders machines in Japan, with each unit earning an average of $10,000 or $46 per day in 100 yen coins. However, this was insufficient to meet the high demand, prompting Taito to increase production to 25,000-30,000 units per month and increase projections to 400,000 manufactured in Japan by the end of 1979.To meet demand, Taito licenced overseas rights to Midway for distribution outside of Japan. By the end of 1979, an estimated 750,000 Space Invaders machines had been installed worldwide, including 400,000 in Japan, 85,000 in the United Kingdom, and 60,000 in the United States within a year; the game eventually sold 72,000 units in the United States by 1982. By 1979, it had become the all-time best-seller in the arcade game industry.Space Invaders had a long run in Australia, with a level of longevity not matched until Street Fighter II (1991) After quadrupling the system's sales, the 1980 Atari VCS (Atari 2600) version became the first official licencing of an arcade game for consoles and became the first "killer app" for video game consoles.It sold over two million units in its first year of release as a home console game, making it the first title to do so. The game went on to sell 2.5 million copies, then over 4.2 million by the end of 1981, and over 5.6 million by 1982, making it the best-selling Atari 2600 game until the Atari version of Pac-Man was released (1982).By 1983, Space Invaders for the Atari 2600 had sold 6,091,178 cartridges, with an additional 161,051 sold between 1986 and 1990, for a total of more than 6.25 million cartridges sold by 1990.

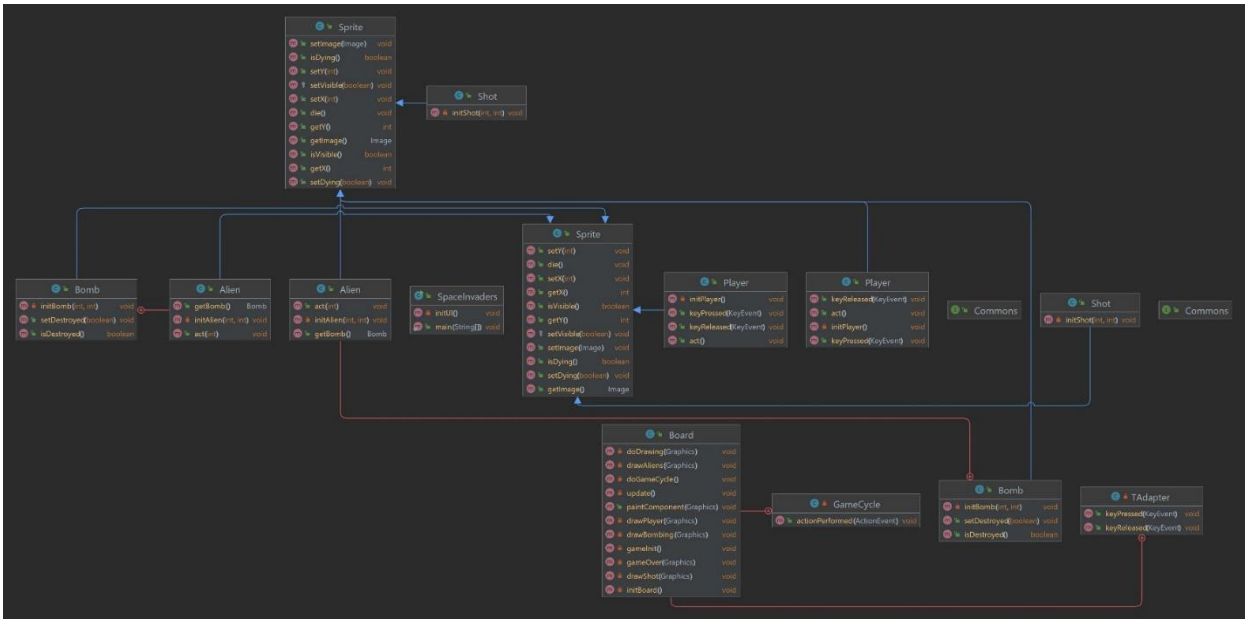
Other official conversions were released for the Atari 8-bit computer line and the Atari 5200 console, with Taito later releasing it for the Nintendo Famicom in Japan in 1985. Space Invaders versions were available for handheld electronic game devices, tabletop dedicated consoles, home computers, watches, and pocket calculators by 1982. Richard Maurer programmed the Atari VCS conversion, while Eric Manghise programmed and Marilyn Churchill animated the Atari 5200 conversion. Adjusted for inflation, total sales of all versions of Space Invaders are estimated to have surpassed $13 billion in gross revenue in 2016, making it the highest-grossing video game of all time.

More than a hundred Space Invaders clones were released for various platforms, including the popular computer games Super Invader (1979) and TI Invaders (1981); the latter was the best-selling game for the TI-99/4A until at least 1982.

CHAPTER 4

UML DIAGRAMS

4.1 CLASS DIAGRAM

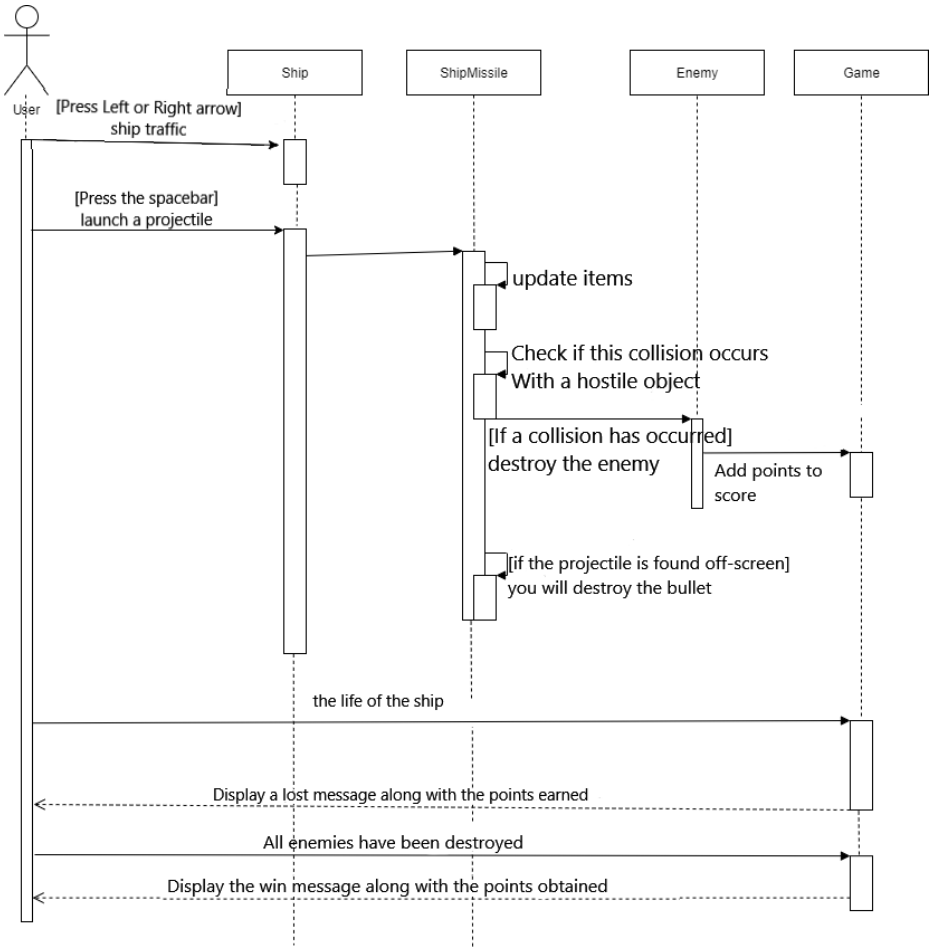


4.2 SEQUENCE DIAGRAM

UML sequence diagrams are used to show how objects interact in a given situation. An important characteristic of a sequence diagram is that time passes from top to bottom: the interaction starts near the top of the diagram and ends at the bottom Sequence diagrams contain the same information as Collaboration diagrams, but emphasize the sequence of the messages instead of the relationships between the objects.

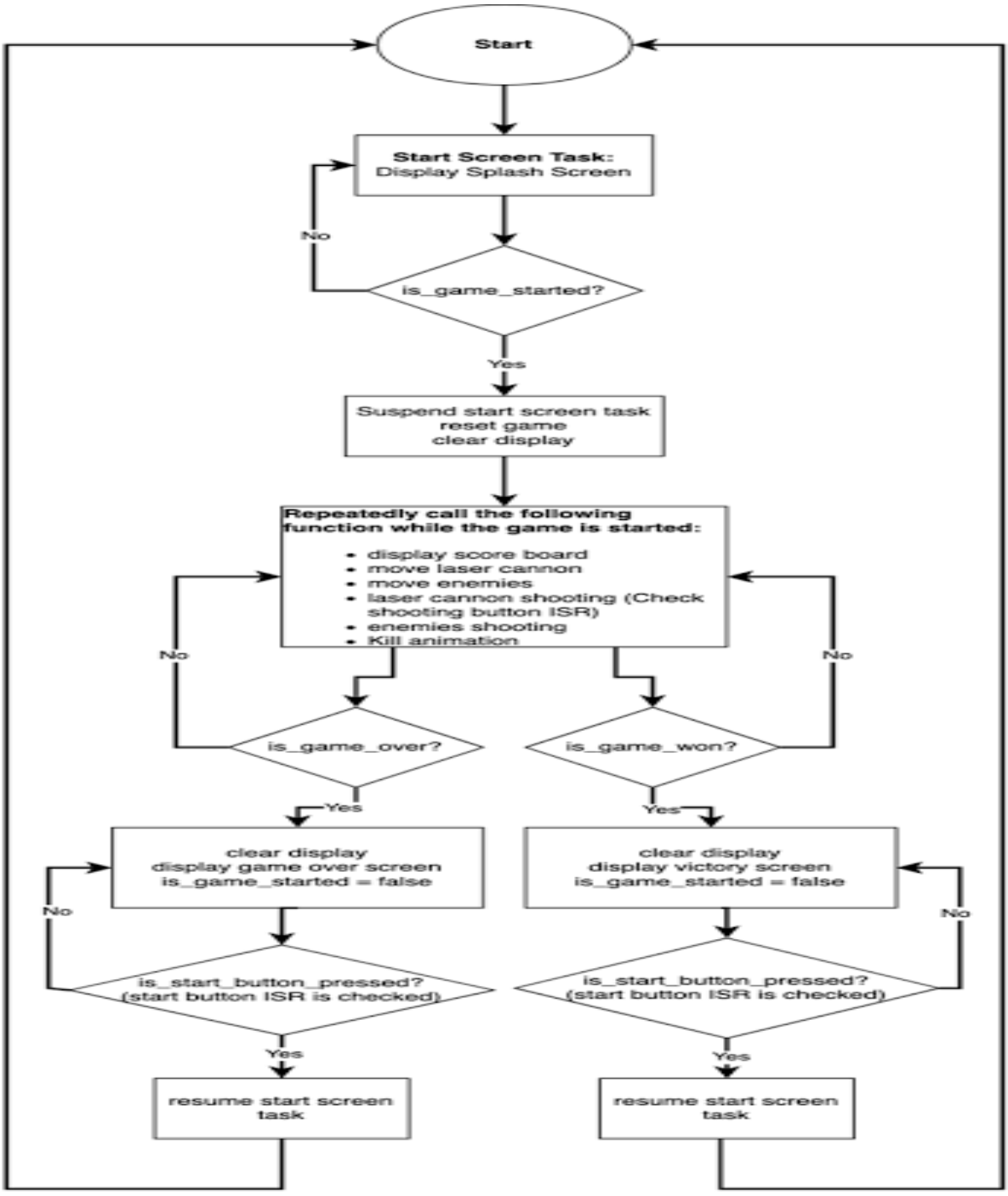
Sequence diagram for voice chat system:

- Client send the request to connect as button connect action performed.
- Server accepts the request, validate it and create the chat handler for client requested the connection.
- Server continues listening on the port for new client requesting for connection.
- Now client writes the text data or voice data to chat handler which broadcast the accepted data.
- All clients accept the data and perform the necessary action, then client and chat handler are destroyed as their work is done.



4.3 DATA FLOW DIAGRAMS

Data flow diagram is also called as ’bubble chart’ is a graphical technique, which is used to represent information flow, and transformers those are applied when data moves from input to output.DFD represents the system requirement clearly and identify transformers those become programming design .DFD consists of series of bubble joint by lines. DFD may be further portioned into different levels to show detail information flow e.g. Level 0, Level 1 etc.



CHAPTER 5
MODULES EXPLANATION

6.1 OVERVIEW

Game Components are the classes and methods that are executed behind the scenes and contribute to the overall functionality of the game. The components are interconnected with one another, and there are many different kinds of interactions between them. The following is a description of the primary components:

The primary class: The main class is the place where all of the game's components are connected, and it also serves as the game's brain. This class contains a representation of the game's graphical user interface (GUI). Additionally, the movement is listened to from this location, and several approaches are utilized for the various functionalities.

There are twenty-four unauthorized users in our copy of Java. These extraterrestrial beings are bombarding the ground with their weapons. When the player fires a rocket, they won't be able to fire another one until the first one either destroys an enemy or reaches the game's highest point value. The player fires their weapon by using the space bar. The aliens disperse their bombs in a haphazard manner. After the explosion caused by the preceding bomb, the subsequent aliens will just detonate their bombs.

6.1 INPUT

The appearance of key events lets you know when the user is actively typing on the keyboard. To be more specific, the component that has the keyboard focus is the one that is responsible for firing key events whenever the user pushes or releases keyboard keys. Please refer to the article titled "How to Use the Focus Subsystem" for further information regarding focus. Instead of using a key listener, you should make use of key bindings in order to create special behaviors to specific keys. In general, you should only respond to events involving the typing of keys unless you have a specific reason to know when the user pushes keys that do not match to characters. Handling key-typed events allows you, for instance, to determine when the user types a Unicode character. This can be accomplished by pressing a single key, such as "a," or by pushing numerous keys in rapid succession. Handling key-pressed events,

on the other hand, is how you find out whether the user hits the F1 key or whether the user touched the '3' key on the number pad.
A component has to have the keyboard focus in order for keyboard events to be fired from it.

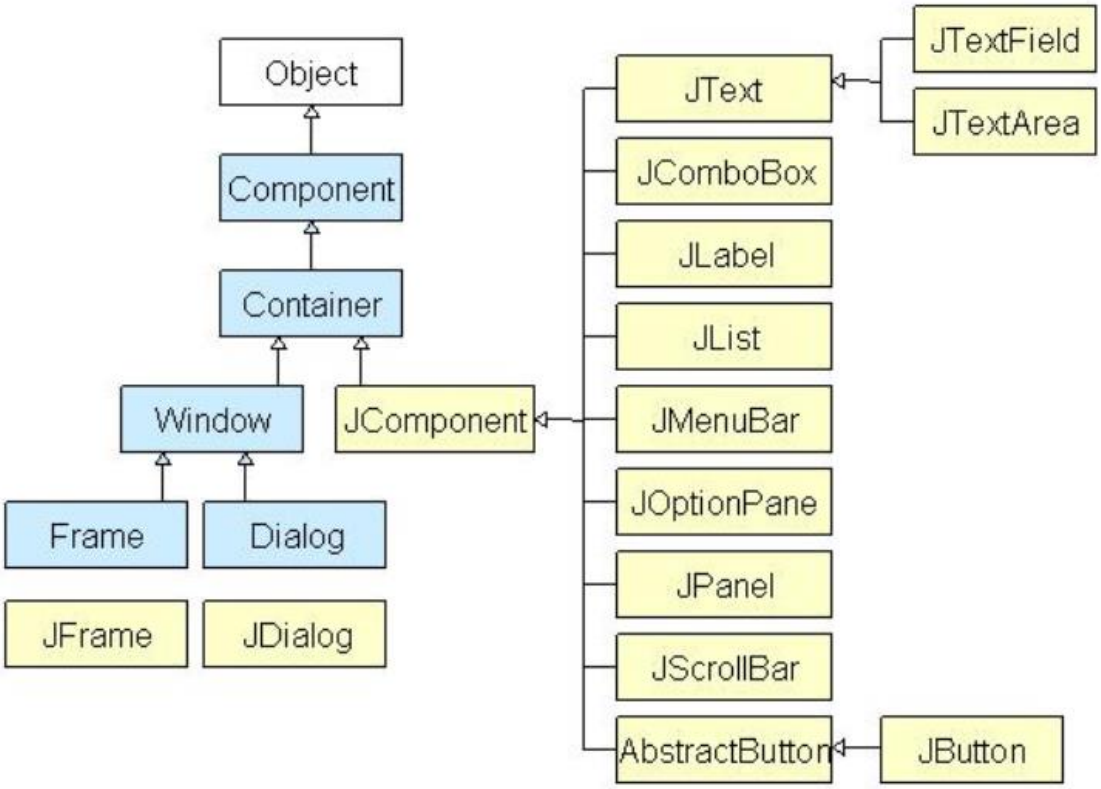
The following actions need to be taken in order to give a component the focus on the keyboard:
Check that the isFocusable method of the component returns the true value. This stage grants the component the ability to become the component of focus. For instance, you can make a JLabel component accept keyboard focus by executing the setFocusable(true) function on the component's label and passing in the value true.
Ensure that the component asks for the focus when it is appropriate to do so. When working with custom components, you should build a mouse listener that, if the component is clicked, calls the requestFocusInWindow method. The focus subsystem utilises focus traversal keys like Tab and Shift Tab to perform its functions. You can use the call component function if you need to stop the focus traversal keys from being consumed in any way. For the component that is responsible for triggering the key events, make sure that setFocusTraversalKeysEnabled is set to false. After then, it is up to your application to handle focus traversal on its own. You also have the option of using the KeyEventDispatcher class in order to pre-listen to all key events before they occur. The information on the focus page pertains to the focus subsystem in great depth. You have the ability to receive specific information regarding a key-pressed event of your choosing. You could, for instance, query a key-pressed event to find out if it was initiated by an action key. The keys labelled Copy, Paste, Page Up, and Undo are all examples of action keys, as are the arrow and function keys. You can also query a key-pressed or key-released event to find out which key was pushed or released to cause the event to be triggered. The vast majority of key events are generated by the normal keyboard; however, the events for certain keys, such as Shift, contain information on whether the user hit the Shift key on the left or right side of the keyboard when they pressed the Shift key. Similarly, the number "2" can be typed using either the normal keyboard or the number pad, depending on your preference.

6.2 GRAPHICS

Swing is an extension of the Abstract Window Toolkit (AWT) as well as a library that is part of the Java Foundation Classes (JFC). In comparison to AWT, the capability provided by Swing is far more advanced. It also includes brand new components, expanded component features, superior event management, and support for drag-and-drop operations. Swing is included in the standard Java distribution and features approximately four times as many User Interface (UI) components as AWT does. The AWT is a constrained implementation that is not nearly capable of supplying the components required for constructing the complex GUIs that are required in today's commercial applications. This is because the needs for today's application GUIs have changed. When compared to the resources offered by Swing, the AWT component set is plagued with quite a few flaws and uses up a significant amount of the system's available resources. The Internet Foundation Classes (also known as IFC) library was initially developed by Netscape for use with Java. Programmers that create graphical user interfaces for commercial applications began to show a lot of interest in its Classes.

Swing is an Application Programming Interface Set (API- Set Of Classes and Interfaces)
Swing is made available for use in the design of graphical user interfaces. The AWT is an extension library that contains Swing (Abstract Window Toolkit) Includes new and upgraded Components that have been increasing the appearance of GUIs while also improving their functionality. Swing is capable of being used to develop stand-alone graphical user interface applications. Additionally available as Servlets and Applets
It uses a model-view based architecture for its design.

The AWT is less cumbersome and more rigid than Swing, which is built on top of the AWT. Swing is built on top of the AWT. Swing is entirely composed of Java code. The Java Swing Components are independent of any particular platform, and Swing Components themselves are quite lightweight. Pluggable appearance and sensations are supported by Swing. In addition, Swing offers additional potent components such as tables, lists, Scrollpanes, Colorchooser, tabbedpanes, and so on.

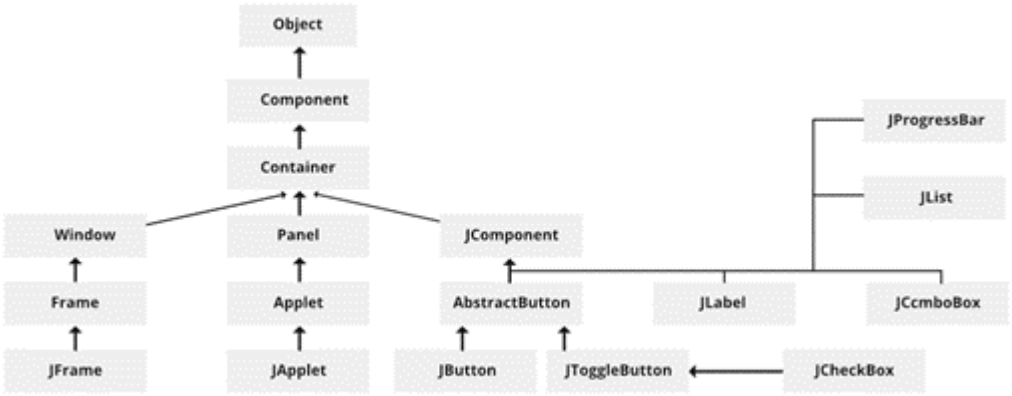


MVC Is Followed By Further Swing

It is a common misconception among programmers that JFC and Swing are the same thing; however, this is not the case. Advanced features like JTable, JTabbedPane, and JScollPane, among others, are included. Since Java is a platform-independent language, it can be run on any client machine. However, the look and feel of the graphical user interface (GUI), which is owned and delivered by the operating system (OS) of the client machine, does not affect an application's GUI that is constructed using Swing components. Components That Are Low in Weight: The development of lightweight components was supported by its AWT beginning with the JDK 1.1 release. In order for a component to be considered lightweight, it must not rely on any system classes that are not based on the Java operating system. The look and feel classes in Java provide support for the individual views of the Swing components. Pluggable Look and Feel: This feature gives the user the ability to change the appearance of the Swing components without having to restart the application. The Swing library provides support for components with a look and feel that is consistent across all platforms, regardless of where the programme is being run. The Swing library offers an

application programming interface (API) that provides a great deal of leeway in deciding how an application's graphical user interface (GUI) will appear.

The Order of the Swing Classes



The Relationship With MVC

A visual component is, in most cases, a combination of the following three unique aspects: The appearance of the component when it is displayed on the screen after being rendered.

The manner in which the component responds to input from the user The information regarding the component's state that is associated with it One component architecture in particular has established a track record through the years as being highly efficient:

- Model-View-Controller, abbreviated to MVC for convenience.

The model is equivalent to the state information that is associated with the component according to the nomenclature of MVC. The view decides how the component will be displayed on the screen, including any features of the view that are affected by the current state of the model. The view also takes into account any other information that is relevant to the view. The controller is responsible for determining how the component behaves in response to the user. Even the most basic components of Swing have features that considerably exceed those of AWT components.

CHAPTER 6
SOURCE CODE

SpaceInvaders.java

```
package com.space;

import java.awt.EventQueue;
import javax.swing.JFrame;

public class SpaceInvaders extends JFrame  {

    public SpaceInvaders() {

        initUI();
    }

    private void initUI() {

        add(new Board());

        setTitle("Space Invaders");
        setSize(Commons.BOARD_WIDTH,
Commons.BOARD_HEIGHT);

        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setResizable(false);
        setLocationRelativeTo(null);
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            var ex = new SpaceInvaders();
            ex.setVisible(true);
        });
    }
}
```

Commons.java

```
package com.space;

public interface Commons {

    int BOARD_WIDTH = 358;
    int BOARD_HEIGHT = 350;
    int BORDER_RIGHT = 30;
    int BORDER_LEFT = 5;

    int GROUND = 290;
    int BOMB_HEIGHT = 5;

    int ALIEN_HEIGHT = 12;
    int ALIEN_WIDTH = 12;
    int ALIEN_INIT_X = 150;
    int ALIEN_INIT_Y = 5;

    int GO_DOWN = 15;
    int NUMBER_OF_ALIENS_TO_DESTROY = 24;
    int CHANCE = 5;
    int DELAY = 17;
    int PLAYER_WIDTH = 15;
    int PLAYER_HEIGHT = 10;
}
```

Alien.java

```
package com.space.sprite;

import javax.swing.ImageIcon;

public class Alien extends Sprite {

    private Bomb bomb;

    public Alien(int x, int y) {

        initAlien(x, y);
    }

    private void initAlien(int x, int y) {

        this.x = x;
        this.y = y;

        bomb = new Bomb(x, y);

        var alienImg = "src/images/alien.png";
        var ii = new ImageIcon(alienImg);

        setImage(ii.getImage());
    }

    public void act(int direction) {

        this.x += direction;
    }

    public Bomb getBomb() {

        return bomb;
    }

    public class Bomb extends Sprite {

        private boolean destroyed;

        public Bomb(int x, int y) {
```

```
        initBomb(x, y);
    }

    private void initBomb(int x, int y) {

        setDestroyed(true);

        this.x = x;
        this.y = y;

        var bombImg = "src/images/bomb.png";
        var ii = new ImageIcon(bombImg);
        setImage(ii.getImage());
    }

    public void setDestroyed(boolean destroyed) {

        this.destroyed = destroyed;
    }

    public boolean isDestroyed() {

        return destroyed;
    }
  }
}
```

Player.java

```
package com.space.sprite;

import com.space.Commons;

import javax.swing.ImageIcon;
import java.awt.event.KeyEvent;

public class Player extends Sprite {

    private int width;

    public Player() {

        initPlayer();
    }

    private void initPlayer() {

        var playerImg = "src/images/player.png";
        var ii = new ImageIcon(playerImg);

        width = ii.getImage().getWidth(null);
        setImage(ii.getImage());

        int START_X = 270;
        setX(START_X);

        int START_Y = 280;
        setY(START_Y);
    }

    public void act() {

        x += dx;

        if (x <= 2) {

            x = 2;
        }

        if (x >= Commons.BOARD_WIDTH - 2 * width) {
```

```
            x = Commons.BOARD_WIDTH - 2 * width;
        }
    }

    public void keyPressed(KeyEvent e) {

        int key = e.getKeyCode();

        if (key == KeyEvent.VK_LEFT) {

            dx = -2;
        }

        if (key == KeyEvent.VK_RIGHT) {

            dx = 2;
        }
    }

    public void keyReleased(KeyEvent e) {

        int key = e.getKeyCode();

        if (key == KeyEvent.VK_LEFT) {

            dx = 0;
        }

        if (key == KeyEvent.VK_RIGHT) {

            dx = 0;
        }
    }
}
```

Shot.java

```
package com.space.sprite;


import javax.swing.ImageIcon;


public class Shot extends Sprite {
    public Shot() {
    }
    public Shot(int x, int y) {
        initShot(x, y);
    }


    private void initShot(int x, int y) {
        var shotImg = "src/images/shot.png";
        var ii = new ImageIcon(shotImg);
        setImage(ii.getImage());
        int H_SPACE = 6;
        setX(x + H_SPACE);
        int V_SPACE = 1;
        setY(y - V_SPACE);
    }
}
```

Sprite.java

```
package com.space.sprite;

import java.awt.Image;

public class Sprite {

    private boolean visible;
    private Image image;
    private boolean dying;

    int x;
    int y;
    int dx;

    public Sprite() {

        visible = true;
    }

    public void die() {

        visible = false;
    }

    public boolean isVisible() {

        return visible;
    }

    protected void setVisible(boolean visible) {

        this.visible = visible;
    }

    public void setImage(Image image) {

        this.image = image;
    }

    public Image getImage() {
```

```
        return image;
    }

    public void setX(int x) {

        this.x = x;
    }

    public void setY(int y) {

        this.y = y;
    }

    public int getY() {

        return y;
    }

    public int getX() {

        return x;
    }

    public void setDying(boolean dying) {

        this.dying = dying;
    }

    public boolean isDying() {

        return this.dying;
    }
}
```

Board.java

```
package com.space;

import com.zetcode.sprite.Alien;
import com.zetcode.sprite.Player;
import com.zetcode.sprite.Shot;

import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.Timer;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Random;

public class Board extends JPanel {

    private Dimension d;
    private List<Alien> aliens;
    private Player player;
    private Shot shot;

    private int direction = -1;
    private int deaths = 0;

    private boolean inGame = true;
    private String explImg = "src/images/explosion.png";
    private String message = "Game Over";

    private Timer timer;


    public Board() {
```

```
        initBoard();
        gameInit();
    }

    private void initBoard() {

        addKeyListener(new TAdapter());
        setFocusable(true);
        d = new Dimension(Commons.BOARD_WIDTH,
Commons.BOARD_HEIGHT);
        setBackground(Color.black);

        timer = new Timer(Commons.DELAY, new GameCycle());
        timer.start();

        gameInit();
    }


    private void gameInit() {

        aliens = new ArrayList<>();

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 6; j++) {

                var alien = new Alien(Commons.ALIEN_INIT_X + 18 * j,
                        Commons.ALIEN_INIT_Y + 18 * i);
                aliens.add(alien);
            }
        }

        player = new Player();
        shot = new Shot();
    }

    private void drawAliens(Graphics g) {

        for (Alien alien : aliens) {

            if (alien.isVisible()) {
```

```
                g.drawImage(alien.getImage(), alien.getX(), alien.getY(),
this);
            }

            if (alien.isDying()) {

                alien.die();
            }
        }
    }

    private void drawPlayer(Graphics g) {

        if (player.isVisible()) {

            g.drawImage(player.getImage(), player.getX(), player.getY(),
this);
        }

        if (player.isDying()) {

            player.die();
            inGame = false;
        }
    }

    private void drawShot(Graphics g) {

        if (shot.isVisible()) {

            g.drawImage(shot.getImage(), shot.getX(), shot.getY(), this);
        }
    }

    private void drawBombing(Graphics g) {

        for (Alien a : aliens) {

            Alien.Bomb b = a.getBomb();

            if (!b.isDestroyed()) {

                g.drawImage(b.getImage(), b.getX(), b.getY(), this);
```

```
        }
    }
}

@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);

    doDrawing(g);
}

private void doDrawing(Graphics g) {

    g.setColor(Color.black);
    g.fillRect(0, 0, d.width, d.height);
    g.setColor(Color.green);

    if (inGame) {

        g.drawLine(0, Commons.GROUND,
                Commons.BOARD_WIDTH, Commons.GROUND);

        drawAliens(g);
        drawPlayer(g);
        drawShot(g);
        drawBombing(g);

    } else {

        if (timer.isRunning()) {
            timer.stop();
        }

        gameOver(g);
    }

    Toolkit.getDefaultToolkit().sync();
}

private void gameOver(Graphics g) {

    g.setColor(Color.black);
```

```
        g.fillRect(0, 0, Commons.BOARD_WIDTH,
Commons.BOARD_HEIGHT);

        g.setColor(new Color(0, 32, 48));
        g.fillRect(50, Commons.BOARD_WIDTH / 2 - 30,
Commons.BOARD_WIDTH - 100, 50);
        g.setColor(Color.white);
        g.drawRect(50, Commons.BOARD_WIDTH / 2 - 30,
Commons.BOARD_WIDTH - 100, 50);

        var small = new Font("Helvetica", Font.BOLD, 14);
        var fontMetrics = this.getFontMetrics(small);

        g.setColor(Color.white);
        g.setFont(small);
        g.drawString(message, (Commons.BOARD_WIDTH -
fontMetrics.stringWidth(message)) / 2,
                Commons.BOARD_WIDTH / 2);
    }

    private void update() {

        if (deaths == Commons.NUMBER_OF_ALIENS_TO_DESTROY)
{

            inGame = false;
            timer.stop();
            message = "Game won!";
        }

        // player
        player.act();

        // shot
        if (shot.isVisible()) {

            int shotX = shot.getX();
            int shotY = shot.getY();

            for (Alien alien : aliens) {

                int alienX = alien.getX();
                int alienY = alien.getY();
```

```
            if (alien.isVisible() && shot.isVisible()) {
                if (shotX >= (alienX)
                        && shotX <= (alienX + Commons.ALIEN_WIDTH)
                        && shotY >= (alienY)
                        && shotY <= (alienY +
Commons.ALIEN_HEIGHT)) {

                    var ii = new ImageIcon(explImg);
                    alien.setImage(ii.getImage());
                    alien.setDying(true);
                    deaths++;
                    shot.die();
                }
            }
        }

        int y = shot.getY();
        y -= 4;

        if (y < 0) {
            shot.die();
        } else {
            shot.setY(y);
        }
    }

    // aliens

    for (Alien alien : aliens) {

        int x = alien.getX();

        if (x >= Commons.BOARD_WIDTH -
Commons.BORDER_RIGHT && direction != -1) {

            direction = -1;

            Iterator<Alien> i1 = aliens.iterator();

            while (i1.hasNext()) {

                Alien a2 = i1.next();
```

```
                a2.setY(a2.getY() + Commons.GO_DOWN);
            }
        }

        if (x <= Commons.BORDER_LEFT && direction != 1) {

            direction = 1;

            Iterator<Alien> i2 = aliens.iterator();

            while (i2.hasNext()) {

                Alien a = i2.next();
                a.setY(a.getY() + Commons.GO_DOWN);
            }
        }
    }

    Iterator<Alien> it = aliens.iterator();

    while (it.hasNext()) {

        Alien alien = it.next();

        if (alien.isVisible()) {

            int y = alien.getY();

            if (y > Commons.GROUND - Commons.ALIEN_HEIGHT) {
                inGame = false;
                message = "Invasion!";
            }

            alien.act(direction);
        }
    }

    // bombs
    var generator = new Random();

    for (Alien alien : aliens) {

        int shot = generator.nextInt(15);
```

```
        Alien.Bomb bomb = alien.getBomb();

        if (shot == Commons.CHANCE && alien.isVisible() &&
bomb.isDestroyed()) {
            bomb.setDestroyed(false);
            bomb.setX(alien.getX());
            bomb.setY(alien.getY());
        }
        int bombX = bomb.getX();
        int bombY = bomb.getY();
        int playerX = player.getX();
        int playerY = player.getY();

        if (player.isVisible() && !bomb.isDestroyed()) {

            if (bombX >= (playerX)
                    && bombX <= (playerX +
Commons.PLAYER_WIDTH)
                    &;& bombY >= (playerY)
                    && bombY <= (playerY +
Commons.PLAYER_HEIGHT)) {

                var ii = new ImageIcon(explImg);
                player.setImage(ii.getImage());
                player.setDying(true);
                bomb.setDestroyed(true);
            }
        }

        if (!bomb.isDestroyed()) {

            bomb.setY(bomb.getY() + 1);

            if (bomb.getY() >= Commons.GROUND -
Commons.BOMB_HEIGHT) {

                bomb.setDestroyed(true);
            }
        }
    }
}

private void doGameCycle() {
```

```
        update();
        repaint();
    }

    private class GameCycle implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {

            doGameCycle();
        }
    }
    private class TAdapter extends KeyAdapter {

        @Override
        public void keyReleased(KeyEvent e) {

            player.keyReleased(e);
        }
        @Override
        public void keyPressed(KeyEvent e) {

            player.keyPressed(e);

            int x = player.getX();
            int y = player.getY();

            int key = e.getKeyCode();

            if (key == KeyEvent.VK_SPACE) {

                if (inGame) {

                    if (!shot.isVisible()) {

                        shot = new Shot(x, y);
                    }
                }
            }
        }
    }
}
```

CHAPTER 6
OUTPUT SCREENSHOTS

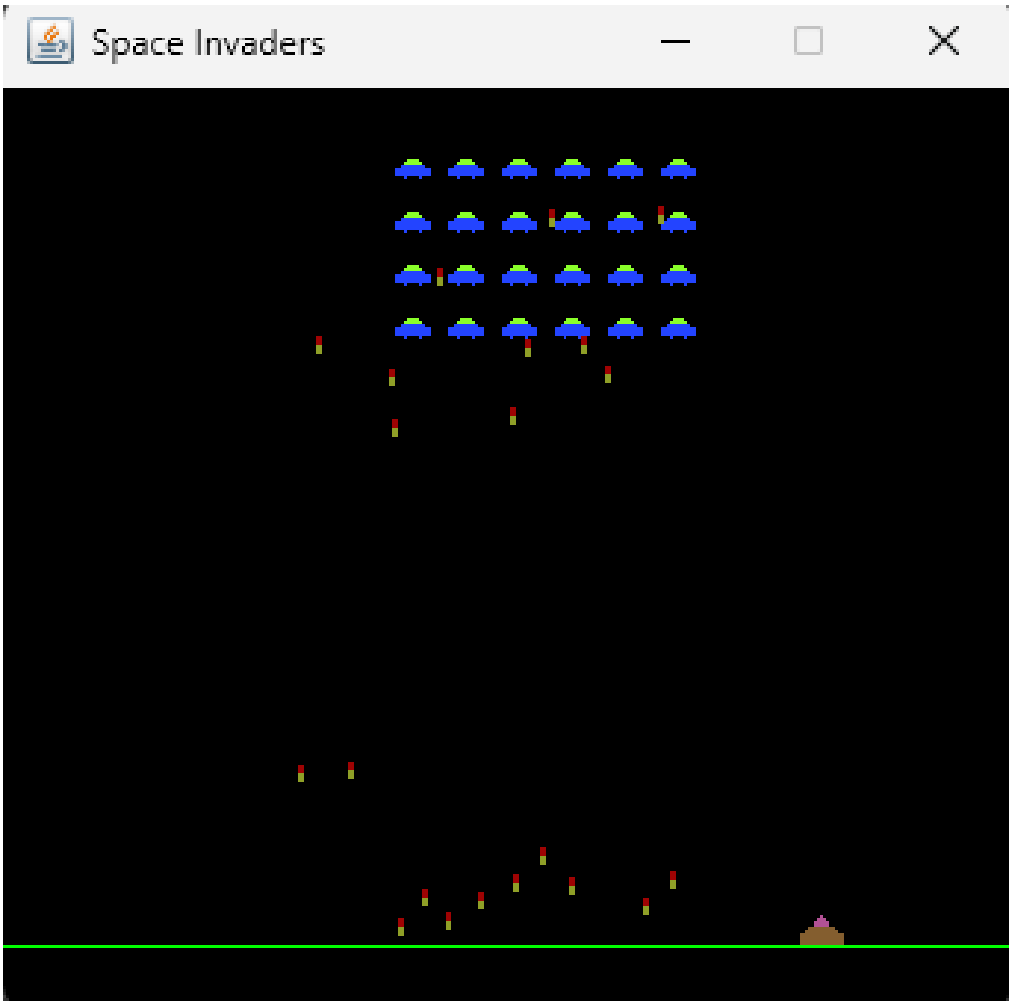


Figure 6.1 (Game Screen)

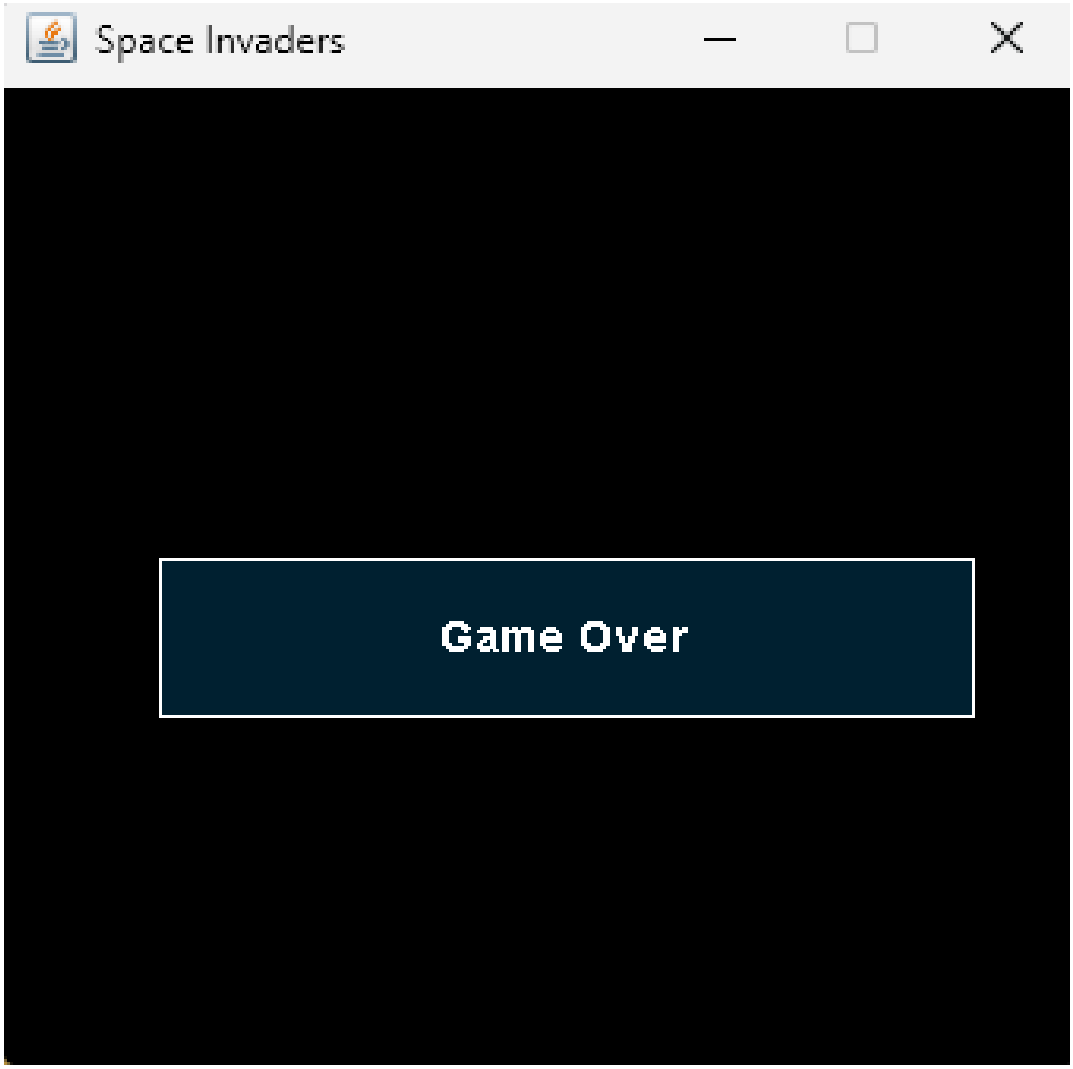


Figure 6.2 (End Game)

CHAPTER 7

CONCLUSION

We gained knowledge of various methods of project management utilized by industry experts during the creation of large-scale projects. Crucial to the success of the Snake game's development was the team's ability to work together and integrate modules that were created independently, based on nothing more than requirement specifications.

We get more excitement, frustration, and satisfaction out of this project than any other. Planning, designing, developing, managing, programming proficiency, socket programming, and a whole lot more are just some of the areas where this aids us. There were many problems with the coding of the Ship and aliens. A lot of systems had to be written in a bunch of different ways before the right one was found. Examples include the use of two different movement methods prior to the final version, with the latter still being imperfect due to the ship's inability to fire multiple missiles at once. The collision detection mechanisms of both the alien and the ship's missiles were problematic.

REFERENCES

[1] M. Gallager and A. Ryan, “Learning to play Pac-Man: an evolutionary, rule-based approach,” Proceedings of IEEE Congress on Evolutionary Computation, pp. 2462–2469, 2016.

[2] "Space Invaders vs. Star Wars", Executive, Southam Business Publications, vol. 24,p. 9, 1982, retrieved April 30, 2011,

[3] Kevin Bowen. "The Gamespy Hall of Fame: Space Invaders". GameSpy. Archived from the original on April 8, 2008. Retrieved January 27, 2010.

[4] J. Muñoz, G. Gutierrez, and A. Sanchis, “Multi-objective evolution for car setup optimization,” UK Workshop on Computational Intelligence (UKCI), 2010.