ALLENHOUSE INSTITUTE OF TECHNOLOGY, ROOMA, KANPUR

A

MINI PROJECT REPORT (KCS554)

On

Picture Puzzle

Submitted for

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering (AI & ML)

Session-2023-24

Odd Semester

Submitted by:

Mohammad Shoaib Khan

2105051530054

Submitted to:

Mr. Hari Mohan Dixit

Dr. A. P. J. Abdul Kalam Technical University, Lucknow (U.P.)

November – 2023

Table of Contents

Abstract

The "Picture Puzzle" project is a dive into making games, focusing on being simple and a way to learn. The project was a deliberate choice to create a game in a short time, about one to two weeks, to understand how game development works. Using C# with the MonoGame framework, the game is a sliding puzzle where players arrange tiles to make a complete picture.

The main goal of the game is to challenge players' thinking and problem-solving skills. While not made for a big audience, the game serves as a personal experiment to learn and have fun. It requires the .NET 6.0 runtime to run, showing the technical side.

In this report, we explore the feasibility study, covering technical, economic, organizational, and operational aspects. We also talk about what hardware and software are needed, how the game was designed, different testing methods, and how security was implemented. We finish by discussing potential future improvements. The "Picture Puzzle" project is a quick but meaningful journey into game development, mixing learning with the joy of creating an interesting game.

1 Introduction

A sliding picture puzzle game is a classic and engaging form of entertainment that challenges players to rearrange pieces of an image to form a complete picture. Typically, the image is divided into a grid of squares, and only one square is empty. Players must slide the pieces into the empty space, one at a time, until the picture is successfully reconstructed. The challenge lies in finding the right sequence of moves to solve the puzzle.

"Picture Puzzle" takes this timeless concept and brings it into the digital realm. Using the C# programming language and the MonoGame framework, we crafted a virtual sliding picture puzzle game that combines simplicity with cognitive engagement. Players navigate through the challenge of rearranging tiles to reveal a complete image, fostering problem-solving skills and providing an enjoyable experience.

The motivation behind creating "Picture Puzzle" was twofold. First and foremost, we sought a hands-on exploration into the world of game development. The project served as a practical exercise to understand the intricacies of creating interactive and enjoyable digital experiences. Second, the decision to develop a sliding picture puzzle game was rooted in its simplicity, making it an ideal project to complete within a short timeframe of one to two weeks.

The primary purpose of "Picture Puzzle" is to offer players an entertaining yet intellectually stimulating experience. By challenging users to think critically and strategize their moves, the game aims to enhance problem-solving skills in a fun and interactive manner. While not designed for widespread publication, the project stands as a personal experiment, providing valuable insights into the process of game development and offering a glimpse into the exciting possibilities that digital creations can unfold.

2 Literature Review

2.1 Aim

The aim of this study is to contribute to the evolving landscape of puzzle game development, with a specific focus on the creation of a sliding picture puzzle game titled "Picture Puzzle." The study seeks to gain practical insights into game development methodologies, algorithmic approaches for puzzle solvability, and user interaction design. By undertaking the development of "Picture Puzzle," the aim is to deepen the understanding of coding practices, asset management, and the overall development lifecycle, leveraging C# and the MonoGame framework. Furthermore, the study aims to implement and evaluate the 8-bit puzzle algorithm to determine the solvability of randomly generated puzzles. This aim extends to exploring various user interaction methods, including mouse, keyboard, and Xbox controller inputs, to enhance accessibility and engagement. Additionally, the study aims to implement customization features, such as board selection and difficulty levels, to understand their impact on user satisfaction and gaming experience. Ultimately, the overarching aim is to contribute to the knowledge base in game development and puzzle gaming through the creation and analysis of the "Picture Puzzle" game.

2.2 Objective

The objectives of this study are multifaceted and align with the overarching aim. Firstly, the study aims to gain practical insights into the development lifecycle of a game, delving into coding practices and asset management using C# and the MonoGame framework. It seeks to implement and evaluate the 8-bit puzzle algorithm, contributing to the field of puzzle solvability algorithms. User interaction design is a crucial objective, exploring and implementing various methods like mouse, keyboard, and Xbox controller inputs to enhance user engagement and accessibility. The implementation of customization features, allowing users to choose different boards and difficulty levels, serves to understand their impact on user satisfaction. Through these objectives, the study aims to provide a comprehensive exploration of game development, algorithmic approaches, and user interaction design within the context of sliding picture puzzle games.

2.3 Previous Works

Historical Evolution of Sliding Puzzle Games:

The historical evolution of sliding puzzle games provides a rich context for understanding the genre's foundations. Notably, classics like the 15-puzzle, dating back to the 19th century, showcase the early development of sliding puzzles. These historical examples illuminate the enduring appeal of rearranging tiles to form a complete image, laying the groundwork for subsequent innovations.

Algorithmic Approaches to Puzzle Solvability:

Research on algorithmic approaches to puzzle solvability has significantly contributed to the field. Parity-based algorithms, exemplified by the 8-bit puzzle algorithm, focus on analyzing tile arrangements to determine a puzzle's solvability. These approaches ensure that puzzles are not only challenging but also guarantee a satisfying and achievable experience for players. Furthermore, optimized solving strategies underscore the importance of efficient algorithms tailored to specific puzzle types, enriching our understanding of effective puzzle-solving methodologies.

User Interaction Design in Puzzle Games:

Studies on user interaction design within the realm of puzzle games explore the integration of diverse input methods. The inclusion of options such as mouse, keyboard, and controller support is fundamental for enhancing accessibility and accommodating varied player preferences. The evolution of input methods in puzzle games has been pivotal in creating inclusive and engaging gaming experiences.

Customization Features for Enhanced Engagement:

The integration of customization features in puzzle games has been a notable area of exploration. Previous works have demonstrated the positive impact of allowing players to choose different boards, difficulty levels, or themes. Similar to the "Picture Puzzle" project, these customization features contribute to increased player engagement, satisfaction, and the overall replay value of puzzle games. Understanding the success of customization in previous works provides valuable insights for the effective implementation of similar features in the current project.

In summary, the exploration of historical developments, algorithmic advancements, user interaction design, and customization features in previous works establishes a comprehensive foundation for the subsequent development and analysis of the "Picture Puzzle" game.

3 Methodology

3.1 Feasibility Study

The feasibility study demonstrates that "Picture Puzzle" is not only technically viable but also economically and operationally feasible for a small-scale, educational project. The use of open-source tools and a well-defined team structure contribute to the overall success of the feasibility assessment. The project aligns with its objectives, providing an effective platform for skill development and knowledge acquisition in the field of game development.

3.1.1 Technical Feasibility

The choice of C# programming language and the MonoGame framework proved technically sound for developing the "Picture Puzzle" game. C# offers readability and versatility, while MonoGame provides a robust platform for graphics rendering, user input handling, and overall game management.

Ensuring compatibility with the .NET 6.0 runtime was essential. The decision was validated by the widespread use of .NET and the availability of resources for seamless integration.

3.1.2 Economic Feasibility

The project's feasibility was enhanced by the simplicity of the game and the availability of open-source tools such as MonoGame. Development costs were kept minimal, aligning with the project's short timeline.

While "Picture Puzzle" is not designed for commercial distribution, the knowledge gained from the project serves as a valuable return on investment, contributing to the developers' skill enhancement.

3.1.3 Organizational Feasibility

The small-scale nature of the project allowed for an efficient team structure. Roles and responsibilities were well-defined, and communication was streamlined.

The project provided an excellent opportunity for team members to acquire new skills in game development, enhancing the overall competency of the team.

3.1.4 Operational Feasibility

The game's design ensured smooth integration with the .NET 6.0 runtime, minimizing operational challenges. The lightweight nature of the game also ensured ease of deployment.

The user interface was designed with simplicity in mind, ensuring an intuitive and user-friendly experience for players.

3.2 Requirement Specification

3.2.1 Hardware Requirements

This is the minimum configuration for the users to play the game effectively.

- Processor: Any modern processor with dual-core capabilities or above.
- Memory (RAM): Minimum 2 GB RAM.
- Storage: At least 100 MB of free disk space.

3.2.2 Software Requirements

These are the required software for running the game.

- Operating System: Windows (Windows 7 or above) or Linux (Fedora 35 or above)
- .NET 6.0 Runtime: Required for running the "Picture Puzzle" game.
- Development Environment: (For future development) Visual Studio or any C# development environment compatible with MonoGame.

3.3 Technology Used

3.3.1 C# Programming Language

C# (pronounced C sharp) is a versatile and modern programming language developed by Microsoft. It is particularly well-suited for developing Windows applications, making it an excellent choice for creating the "Picture Puzzle" game. C# is known for its readability, object-oriented features, and extensive library support, allowing developers to write clean and efficient code. With a syntax similar to other C-style languages, C# facilitates rapid development and provides a strong foundation for creating interactive and user-friendly applications.

3.3.2 MonoGame Framework

MonoGame is an open-source framework used for building cross-platform games. It is based on Microsoft's XNA framework and allows developers to write games using C#. MonoGame simplifies the game development process by providing a set of reusable and customizable components for tasks like graphics rendering, input handling, and audio processing. Its cross-platform nature enables developers to deploy games on various platforms, including Windows, macOS, Linux, Android, and iOS. The framework's active community and continuous development make it a valuable tool for game developers looking to create engaging and multi-platform experiences.

3.3.3 .NET 6.0 Runtime

.NET is a free, open-source, cross-platform framework developed by Microsoft for building various types of applications, including web, desktop, and mobile. The .NET 6.0 runtime is a crucial component for running the "Picture Puzzle" game. It provides the necessary libraries and runtime environment for executing .NET applications. With performance improvements

and new features, .NET 6.0 enhances the overall development experience, contributing to the efficiency and responsiveness of the game.

3.3.4 Development Tools

The development of the "Picture Puzzle" game was facilitated by a set of powerful and versatile tools tailored to the project's needs. JetBrains Rider, a cross-platform integrated development environment (IDE), took the lead in providing a comprehensive environment for C# coding. Rider's robust features, including advanced code analysis, debugging capabilities, and seamless integration with MonoGame, ensured an efficient development workflow.

Inkscape, an open-source vector graphics editor, played a pivotal role in crafting scalable visual assets for the game. With its precise tools for creating vector graphics, Inkscape enabled the design of logos, icons, and other visual elements, contributing to the aesthetic appeal of the "Picture Puzzle" game.

GIMP (GNU Image Manipulation Program), a raster graphics editor, complemented Inkscape by offering powerful tools for editing and manipulating raster images. Its versatile features, including image retouching and composition tools, added finesse to the graphical elements of the game.

The combined use of JetBrains Rider, Inkscape, and GIMP formed a dynamic trio, addressing the coding and visual design aspects of the project. This toolset streamlined the development process, fostering collaboration and creativity, and ultimately culminating in the creation of the "Picture Puzzle" game.

Git, a distributed version control system, and GitHub, a web-based platform for hosting and collaborating on Git repositories, were instrumental in managing the project's source code. Git facilitated version control, allowing for collaborative development and tracking changes over time. GitHub served as a centralized repository, enabling seamless collaboration among team members and providing a platform for issue tracking and project management. The use of Git and GitHub enhanced code integrity, version tracking, and collaborative workflows throughout the development of the "Picture Puzzle" game. The source code of the game is available at “github.com/khanshoaib3/PicturePuzzle”.

3.4 Module Description

3.4.1 Game Logic Module

The Game Logic Module serves as the nucleus of the "Picture Puzzle" game, orchestrating the dynamic elements that engage players in an immersive puzzle-solving experience. The primary responsibility lies in managing the game state, where user input is processed, moves are evaluated, and the underlying rules of the sliding picture puzzle are enforced. This module operates predominantly within the Update() methods, ensuring a seamless flow of logic that governs tile movement, puzzle completion checks, and the overall interactive dynamics of the gameplay. By intricately coordinating these elements, the Game Logic Module plays a pivotal role in delivering a captivating and intellectually stimulating gaming experience.

3.4.2 Graphics Rendering Module

The Graphics Rendering Module, leveraging the capabilities of the MonoGame framework, breathes life into the visual components of the "Picture Puzzle" game. Through the integration of captivating visuals, this module ensures players are immersed in an aesthetically pleasing and interactive environment. Graphics rendering encompasses the display of images, animations, and the overall user interface, creating a seamless and engaging visual journey. This module primarily operates within the Draw() methods, where every frame comes to life, contributing to the overall allure and responsiveness of the "Picture Puzzle" game. As users interact with the sliding puzzle, the Graphics Rendering Module dynamically adapts, providing a visually rich and satisfying experience.

3.5 Algorithm Used

To ascertain the solvability of the randomly generated jumbled picture, the 8-puzzle algorithm was employed. This algorithm, based on principles of puzzle parity, examines the arrangement of tiles to determine whether the puzzle has a valid solution.

The 8-puzzle algorithm operates by assessing the number of inversions in the puzzle configuration. An inversion occurs when a tile precedes another tile with a lower numerical value. By calculating the parity of inversions, the algorithm establishes whether the puzzle can be solved from its given state.

This solvability check ensures that the generated puzzles are conducive to successful completion, contributing to a positive and engaging user experience.

3.6 Gameplay Mechanics

The "Picture Puzzle" offers a versatile and engaging gameplay experience, allowing players to interact with the puzzle through multiple input methods:

Keyboard Controls:

Players can navigate the puzzle using keyboard controls, providing a tactile and efficient way to rearrange the picture. The arrow keys serve as the primary input for moving the empty block, strategically rearranging the adjacent picture tiles. This method offers precision and speed, allowing players to solve the puzzle with ease.

Mouse Interaction:

For a more intuitive and hands-on approach, the game supports mouse interaction. Players can click on an adjacent block to move it into the empty space, dynamically reshaping the puzzle. The mouse interaction adds a visual and interactive dimension to the gameplay, catering to players who prefer a point-and-click experience.

Xbox Controller Support:

The "Picture Puzzle" extends its accessibility by incorporating Xbox controller support. Players can utilize the D-pad on the controller to move the empty block, providing a console-like gaming experience. This feature enhances the game's versatility, allowing players to choose their preferred input method and enjoy the puzzle-solving process seamlessly.

Whether it's the precision of keyboard controls, the hands-on feel of mouse interaction, or the console experience with an Xbox controller, the gameplay mechanics of the "Picture Puzzle" cater to a diverse audience. This adaptability ensures that players can enjoy the game in a way that suits their preferences and enhances their overall gaming experience.

4 Result Analysis

4.1 Maintenance

Code Modularity and Maintainability:

Ensuring the long-term viability of the "Picture Puzzle" game was a foundational consideration during development. The implementation adheres to Object-Oriented Programming (OOP) principles, emphasizing code modularity and maintainability. By organizing the code into modular units, each encapsulating specific functionalities, the project becomes more scalable and easier to maintain.

The use of OOP concepts, such as classes and encapsulation, facilitates a clear separation of concerns. This modular structure not only enhances readability but also simplifies future updates and modifications. Each class represents a distinct aspect of the game, promoting a clean and understandable codebase.

Continuous Code Review and Refinement:

Maintenance involves an ongoing commitment to code review and refinement. Regularly assessing the codebase for improvements and addressing any identified issues ensures the game remains efficient, reliable, and adaptable. Adopting an iterative approach to development allows for continuous enhancements, bug fixes, and the incorporation of new features.

In summary, the maintenance strategy for the "Picture Puzzle" game revolves around the principles of modularity, security, and continuous improvement. The implementation of OOP concepts lays the groundwork for a codebase that is both scalable and sustainable, promoting ease of maintenance throughout the game's lifecycle.

4.2 Screenshots

4.2.1 Pages

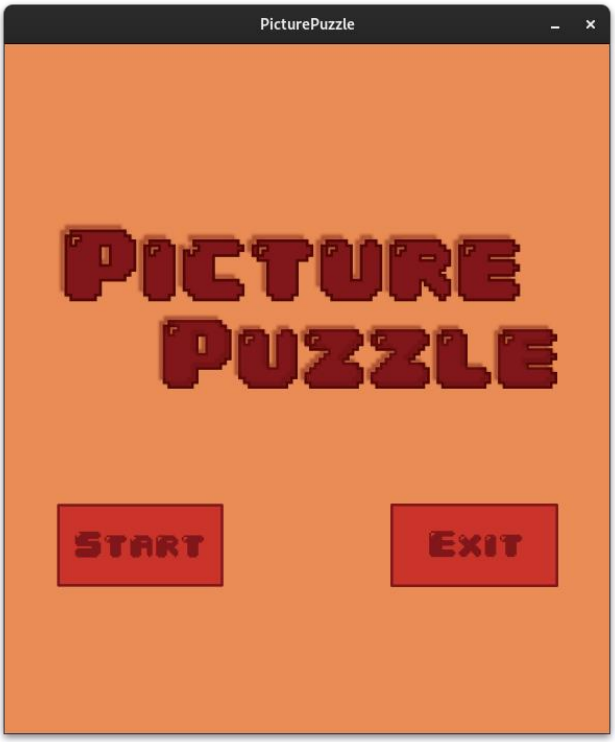


Figure 4-1Title Page

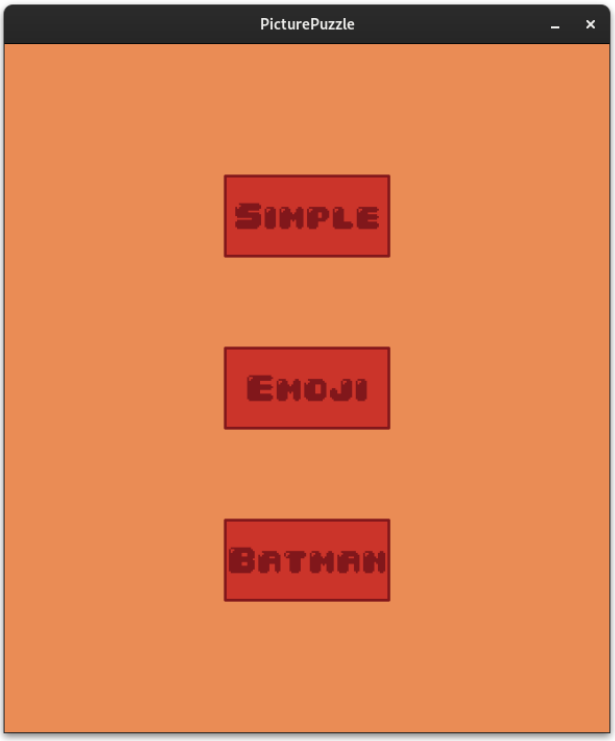


Figure 4-2 Board Selection Page

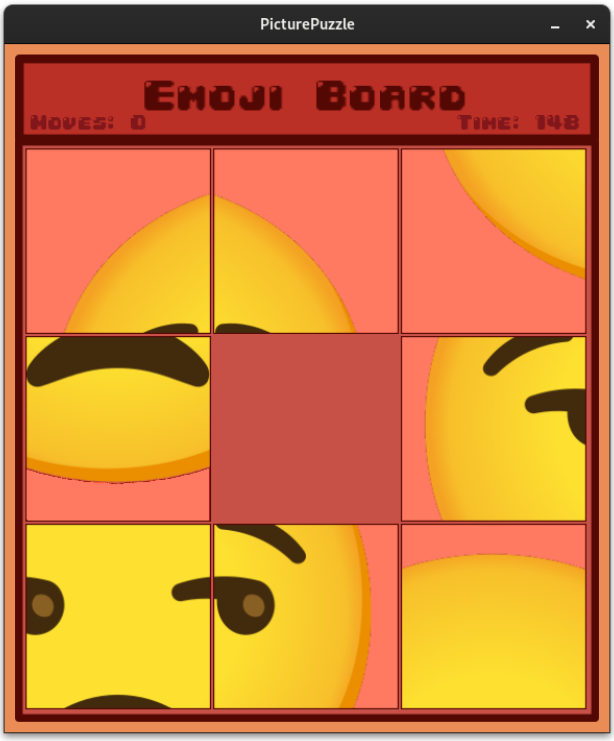


Figure 4-3 Gameplay Page

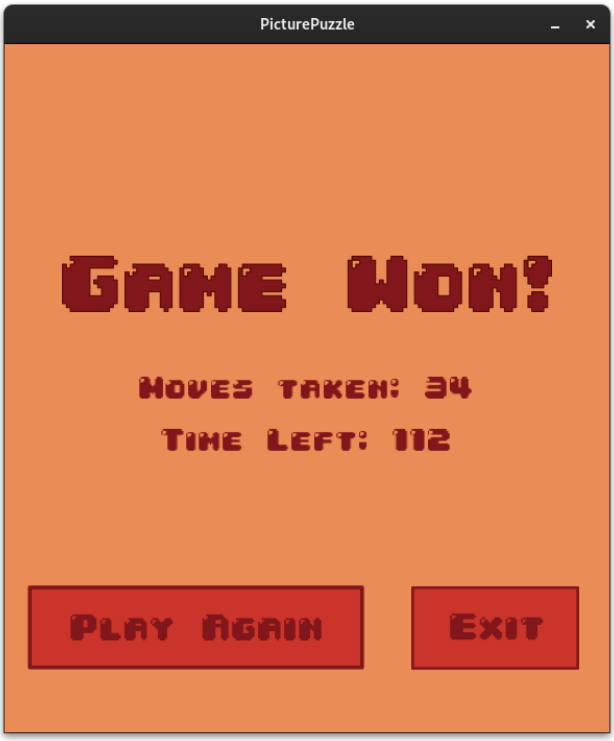


Figure 4-4 End Page

4.2.2 Board Types

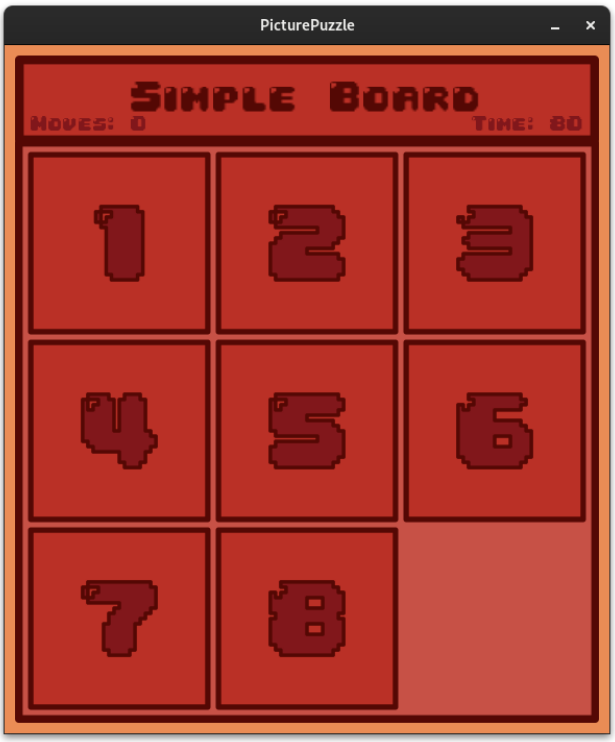


Figure 4-5 Simple Board

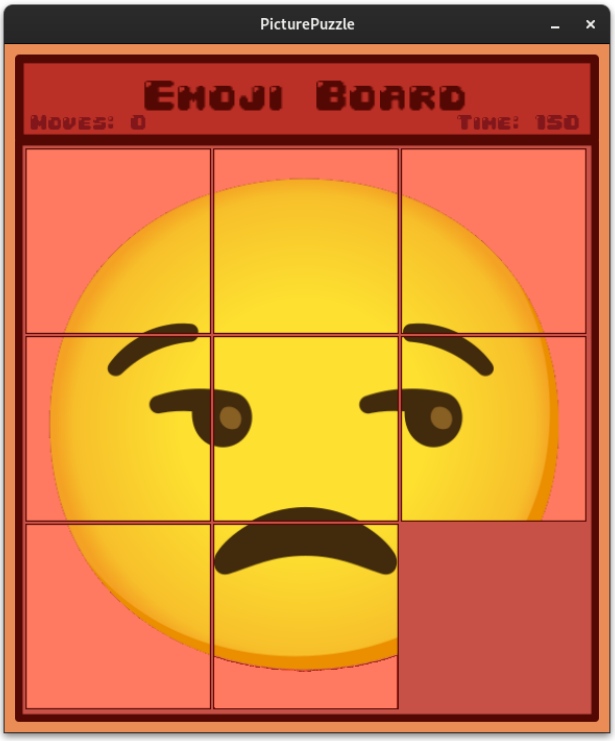


Figure 4-6 Emoji Board

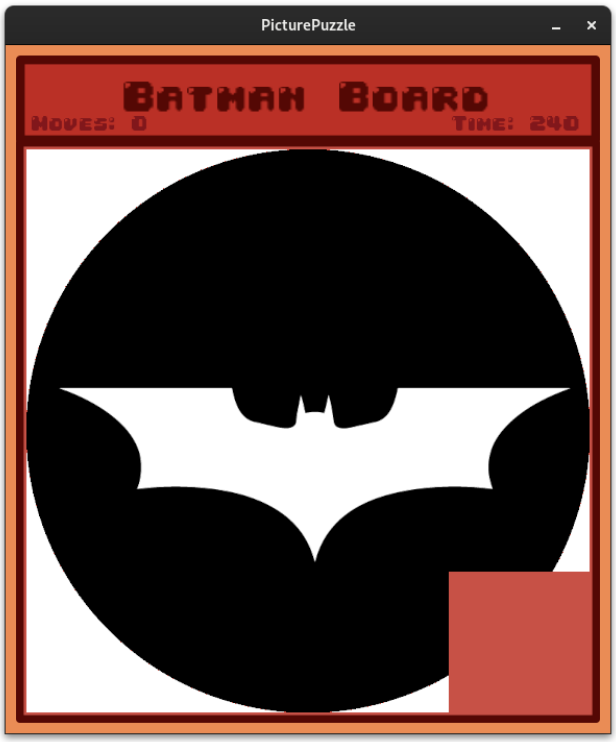


Figure 4-7 Batman Board

5 Conclusion

The development journey of the "Picture Puzzle" game has been a captivating exploration into the field of game development, encompassing challenges, triumphs, and invaluable learning experiences. From the initial conceptualization to the implementation phase, the project was driven by the dual objectives of creating a fun, interactive game and delving into the intricacies of game development.

The choice of technologies, including C#, MonoGame, and JetBrains Rider, laid a solid foundation for the project. Leveraging Inkscape and GIMP for asset creation added a visually appealing dimension to the game. The incorporation of Git and GitHub facilitated collaborative development, ensuring version control and efficient project management.

The modular design of the code, grounded in Object-Oriented Programming principles, not only facilitated a seamless development process but also enhances the game's maintainability. Each module, thoughtfully crafted and encapsulated, contributes to a codebase that is not only functional but also adaptable to future enhancements.

The feasibility study underscored the practicality of the project, emphasizing technical, economic, organizational, and operational aspects. The absence of a database-focused component was a deliberate decision, aligning with the game's scope and objectives. The emphasis on code modularity and security measures ensures a resilient foundation for ongoing maintenance.

As the "Picture Puzzle" game takes its place as a testament to the exploration of game development, it stands as more than just a project—it is a canvas of creativity, a showcase of technical acumen, and a repository of lessons learned. This journey invites future endeavours and promises a continual evolution, inspired by the joy of creating interactive digital experiences.

6 Future Scope

The "Picture Puzzle" game, with its foundation in modular code and an engaging gameplay concept, offers a promising platform for future enhancements and expansions. The following points outline potential avenues for further development:

Diverse Picture Boards and Grid Shapes:

The inclusion of more pictures and the exploration of non-square grid shapes can add variety and complexity to the gameplay. Introducing boards with different dimensions and images enhances the replay value, providing users with a broader range of challenges.

Customizable Image Input:

Expanding the game to allow users to use any picture of their choice for the puzzle opens up a realm of personalization. This feature adds a creative dimension, enabling players to transform their favourite images into engaging puzzles. It not only enhances user engagement but also broadens the game's appeal.

Android Support

Extending the game's compatibility to Android platforms opens up a vast and diverse user base. Adapting the "Picture Puzzle" for mobile devices provides players with the flexibility to enjoy the game on-the-go, reaching a wider audience and enhancing accessibility.

Social and Multiplayer Features

Integrating social and multiplayer features can elevate the game's interactive experience. Leaderboards, achievements, and the ability to challenge friends can foster a sense of competition and community among players, enhancing engagement and retention.

Enhanced Graphics and Animation

Continued improvement in graphics and animation can elevate the visual appeal of the game. Implementing more sophisticated graphics and seamless animations contributes to a more immersive and enjoyable gaming experience.

Accessibility Features

Introducing accessibility features, such as adjustable difficulty levels, colour schemes, and audio cues, ensures inclusivity. Making the game accessible to a broader audience enhances its overall impact and user satisfaction.

Cross-Platform Compatibility

Exploring cross-platform compatibility beyond Android, such as iOS and web platforms, extends the reach of the "Picture Puzzle" game. This adaptability ensures that users can enjoy the game across various devices and operating systems.

The future scope for the "Picture Puzzle" game is rich with possibilities, offering avenues for innovation, personalization, and broader accessibility. By embracing these potential enhancements, the game can evolve into a more dynamic and feature-rich experience, captivating a diverse audience and ensuring its continued relevance in the realm of casual and engaging puzzle games.

7 References

- C# Documentation - https://learn.microsoft.com/en-us/dotnet/csharp/
- MonoGame Documentation – https://docs.monogame.net/
- Inkscape Tutorial - https://inkscape.org/learn/tutorials/
- 8 Puzzle Algorithm - https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/

8 Appendix

8.1 User Manual

8.1.1 Installation

To ensure a smooth setup for the "Picture Puzzle" game, follow the steps outlined below:

1. **Install .NET 6.0 Runtime:**
    a. Before running the game, you need to install the .NET 6.0 runtime. Download and install the runtime from the official .NET website:
        i. For Windows - dotnet.microsoft.com/download/dotnet/6.0/runtime
        ii. For Linux - docs.microsoft.com/en-us/dotnet/core/install/linux
2. **Download the Game Archive:**
    a. Visit the official GitHub releases page at https://github.com/khanshoaib3/PicturePuzzle/releases/tag/v1.0.0
    b. Download the portable archive corresponding to your operating system (Windows or Linux).
3. **Extract the Archive:**
    a. Once the download is complete, extract the contents of the downloaded archive to a location of your choice.
4. **Launch the Game:**
    a. For Windows: Run PicturePuzzle.exe
    b. For Linux: Open a terminal, navigate to the extracted folder, and run ./PicturePuzzle

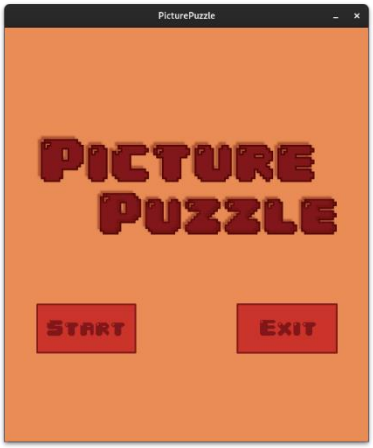
8.1.2 Game Overview

The "Picture Puzzle" game is composed of four distinct pages, each contributing to a seamless and engaging gaming experience.

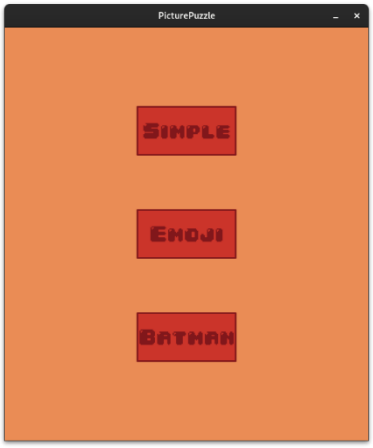
Title Page:

Upon launching the game, you'll be greeted by the Title Page, featuring the game's logo and two prominent buttons:

1. Start:
    - Initiates the game and proceeds to the Board Selection Page.
2. Exit:
    - Exits the game.



Board Selection Page:

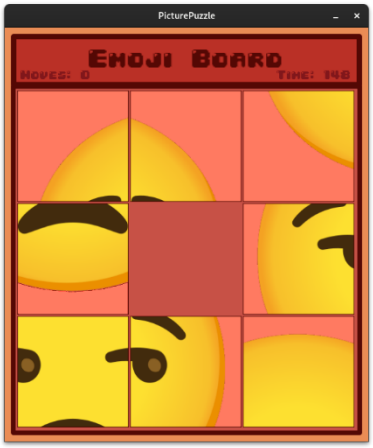


Once you click the "Start" button, you'll transition to the Board Selection Page. Here, you have the opportunity to choose from three exciting boards/pictures:

1. Simple Board:
    - A classic puzzle with a straightforward image.
2. Emoji Board:
    - A playful puzzle featuring emoji images.
3. Batman Logo Board:
    - An adventurous puzzle showcasing the iconic Batman logo.

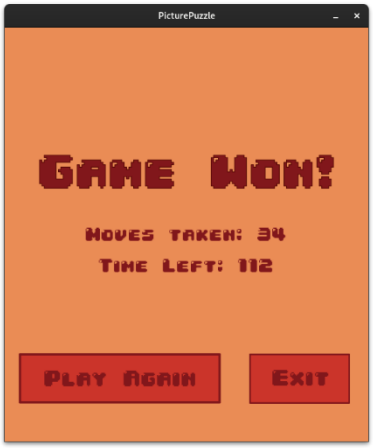
Select your preferred board to customize your gaming experience.

Gameplay Page:



After selecting a board, you'll enter the Gameplay Page, where the main puzzle-solving action takes place. Use your chosen input method (mouse, keyboard, or Xbox controller) to rearrange the jumbled picture into the correct format. This is where your puzzle-solving skills come into play.6

End Page:



Upon completing the puzzle successfully or if the challenge proves too formidable, you'll be directed to the End Page. This page acknowledges your victory or provides a moment to reflect on the game. Celebrate your triumph or strategize for your next attempt.

8.1.3 Getting Started

Mouse Controls:

- Click:
    - Select an adjacent block to move it into the empty space.

Keyboard Controls:

- Arrow Keys:
    - Move the empty block in the corresponding direction (up, down, left, right).

Xbox Controller Support:

- D-pad:
    - Navigate and move the empty block with the Xbox controller.

8.1.4 Gameplay

- **Objective:**
    - Rearrange the picture tiles to reconstruct the complete image.
- **Controls:**
    - Choose your preferred input method (mouse, keyboard, or Xbox controller) to manipulate the tiles and solve the puzzle.
- **Winning:**
    - Successfully arrange all tiles to reveal the complete image within the provided time, marking your victory in the game.

8.1.5 Troubleshooting

- **Performance Issues:**
    - Ensure that your system meets the minimum requirements for optimal performance.
- **Controls Not Responding:**
    - Check the connectivity of your input devices to troubleshoot unresponsive controls.

8.2 Source Code

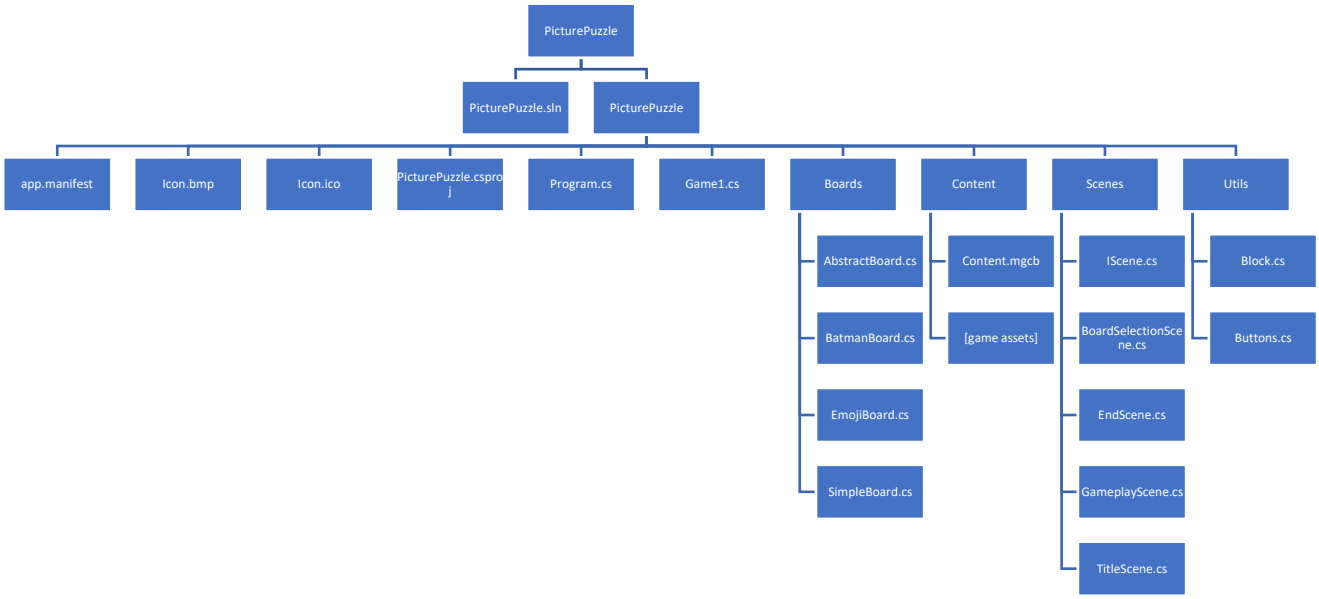


Figure 8-1. Project Hierarchy

Auto-generated Files:

- app.manifest
- Icon.bmp
- Icon.ico
- Program.cs
- PicturePuzzle.csproj
- PicturePuzzle.sln

Game1.cs –

```
using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;


namespace PicturePuzzle;


public class Game1 : Game

{
```

```
// ReSharper disable once NotAccessedField.Local
private GraphicsDeviceManager _graphics;
private SpriteBatch _spriteBatch;
public IScene CurrentScene;

public Game1()
{
    _graphics = new GraphicsDeviceManager(this);
    _graphics.PreferredBackBufferWidth = 580;
    _graphics.PreferredBackBufferHeight = 660;
    Content.RootDirectory = "Content";
    IsMouseVisible = true;
}

protected override void LoadContent()
{
    _spriteBatch = new SpriteBatch(GraphicsDevice);
    CurrentScene = new TitleScene(this);
}

protected override void Update(GameTime gameTime)
{
    CurrentScene?.Update(gameTime, _graphics);

    base.Update(gameTime);
}

protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(new Color(234, 140, 85));
    _spriteBatch.Begin();
```

```
        CurrentScene?.Draw(_spriteBatch);

        _spriteBatch.End();
        base.Draw(gameTime);
    }
}
```

IScene.cs –

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace PicturePuzzle;

public interface IScene
{
    void Update(GameTime gameTime, GraphicsDeviceManager graphics);
    void Draw(SpriteBatch spriteBatch);
}
```

BoardSelectionScene.cs –

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace PicturePuzzle;

public class BoardSelectionScene : IScene
{
    private Game1 _game1;

    private Button _simpleBoard;
```

```
    private Button _emojiBoard;
    private Button _batmanBoard;

    public int Selected { get; private set; }

    public BoardSelectionScene(Game1 game1)
    {
        _game1 = game1;
        _simpleBoard = new Button("sprites/board_selection_scene/simple_button_unhovered",
            "sprites/board_selection_scene/simple_button_hovered", () => Selected = 1, game1);
        _simpleBoard.X = _game1.Window.ClientBounds.Width / 2 - _simpleBoard.Width / 2;
        _simpleBoard.Y = _game1.Window.ClientBounds.Height / 4 - _simpleBoard.Height / 2;

        _emojiBoard = new Button("sprites/board_selection_scene/emoji_button_unhovered",
            "sprites/board_selection_scene/emoji_button_hovered", () => Selected = 2, game1);
        _emojiBoard.X = _game1.Window.ClientBounds.Width / 2 - _emojiBoard.Width / 2;
        _emojiBoard.Y = _game1.Window.ClientBounds.Height / 2 - _emojiBoard.Height / 2;

        _batmanBoard = new
Button("sprites/board_selection_scene/batman_button_unhovered",
            "sprites/board_selection_scene/batman_button_hovered", () => Selected = 3, game1);
        _batmanBoard.X = _game1.Window.ClientBounds.Width / 2 - _batmanBoard.Width / 2;
        _batmanBoard.Y = (3 * _game1.Window.ClientBounds.Height) / 4 -
_batmanBoard.Height / 2;
    }

    public void Update(GameTime gameTime, GraphicsDeviceManager graphics)
    {
        _simpleBoard.Update();
        _emojiBoard.Update();
        _batmanBoard.Update();
    }
```

```
    public void Draw(SpriteBatch spriteBatch)
    {
        _simpleBoard.Draw(spriteBatch);
        _emojiBoard.Draw(spriteBatch);
        _batmanBoard.Draw(spriteBatch);
    }
}
```

EndScene.cs –

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace PicturePuzzle;

public class EndScene : IScene
{
    private Game1 _game1;
    private readonly SpriteFont _04BFont;
    private readonly bool _hasWon;
    private readonly int _moves;
    private readonly int _timeLeft;

    private Button _playAgainButton;
    private Button _exitButton;

    private Texture2D _gameWonTexture;
    private Texture2D _gameLostTexture;

    public EndScene(Game1 game1, bool hasWon, int moves, int timeLeft)
    {
```

```
        _game1 = game1;
        _hasWon = hasWon;
        _moves = moves;
        _timeLeft = timeLeft;
        _04BFont = _game1.Content.Load<SpriteFont>("fonts/04B_30");
        _gameWonTexture =
_game1.Content.Load<Texture2D>("sprites/end_scene/game_won");
        _gameLostTexture =
_game1.Content.Load<Texture2D>("sprites/end_scene/game_lost");
        int centerX = _game1.Window.ClientBounds.Width / 2;
        _playAgainButton = new Button("sprites/common/play_again_button_unhovered",
"sprites/common/play_again_button_hovered",
            () => game1.CurrentScene = new GameplayScene(game1), game1);
        _playAgainButton.X = centerX - _playAgainButton.Width + 55;
        _playAgainButton.Y = _game1.Window.ClientBounds.Height -
_playAgainButton.Height - 60;
        _exitButton = new Button("sprites/common/exit_button_unhovered",
"sprites/common/exit_button_hovered",
            () => game1.Exit(), game1);
        _exitButton.X = centerX + 100;
        _exitButton.Y = _game1.Window.ClientBounds.Height - _exitButton.Height - 60;
    }

    public virtual void Update(GameTime gameTime, GraphicsDeviceManager graphics)
    {
        _playAgainButton.Update();
        _exitButton.Update();
    }

    // ReSharper disable PossibleLossOfFraction
    public virtual void Draw(SpriteBatch spriteBatch)
    {
        Texture2D logoTexture = _hasWon ? _gameWonTexture : _gameLostTexture;
```

```
        int logoX = _game1.Window.ClientBounds.Width / 2 - logoTexture.Width / 2;
        int logoY = _game1.Window.ClientBounds.Height / 2 - logoTexture.Height / 2 -
(_hasWon ? 100 : 80);
        spriteBatch.Draw(logoTexture, new Vector2(logoX, logoY), Color.White);


        if (_hasWon)
        {
            string movesText = $"Moves taken: {_moves}";
            Vector2 movesTextMiddlePoint = _04BFont.MeasureString(movesText) / 2;
            Vector2 movesTextPos = new Vector2(_game1.Window.ClientBounds.Width / 2,
_game1.Window.ClientBounds.Height / 2);
            spriteBatch.DrawString(_04BFont, movesText, movesTextPos,
                new Color(129, 23, 27), 0, movesTextMiddlePoint, 1.2f, SpriteEffects.None, 0.5f);


            string timeText = $"Time Left: {_timeLeft}";
            Vector2 timeTextMiddlePoint = _04BFont.MeasureString(timeText) / 2;
            Vector2 timeTextPos = new Vector2(_game1.Window.ClientBounds.Width / 2,
_game1.Window.ClientBounds.Height / 2 + 50);
            spriteBatch.DrawString(_04BFont, timeText, timeTextPos,
                new Color(129, 23, 27), 0, timeTextMiddlePoint, 1.2f, SpriteEffects.None, 0.5f);
        }


        _playAgainButton.Draw(spriteBatch);
        _exitButton.Draw(spriteBatch);
    }
}
```

GameplayScene.cs –

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
```

```
namespace PicturePuzzle;

public class GameplayScene : IScene
{
    private Game1 _game1;
    private BoardSelectionScene? _boardSelectionPage;
    private AbstractBoard? _currentBoard;
    private TimeSpan? _pressedTime;

    public GameplayScene(Game1 game1)
    {
        _game1 = game1;
        _boardSelectionPage = new BoardSelectionScene(_game1);
    }

    public virtual void Update(GameTime gameTime, GraphicsDeviceManager graphics)
    {
        if (_boardSelectionPage != null)
        {
            switch (_boardSelectionPage.Selected)
            {
                case 0:
                    break;
                case 1:
                    _currentBoard = new SimpleBoard(_game1, 80);
                    _boardSelectionPage = null;
                    break;
                case 2:
                    _currentBoard = new EmojiBoard(_game1, 150);
                    _boardSelectionPage = null;
```

```
                break;
            case 3:
                _currentBoard = new BatmanBoard(_game1, 240);
                _boardSelectionPage = null;
                break;
        }

        _boardSelectionPage?.Update(gameTime, graphics);
        return;
    }

    if (_currentBoard == null) return;

    _currentBoard.Update(gameTime);
    if (_currentBoard.HasEnded)
    {
        _game1.CurrentScene = new EndScene(_game1, _currentBoard.HasWon,
_currentBoard.Moves, _currentBoard.TimeLeft);
        return;
    }

    HandleInputs(gameTime);
  }

  private void HandleInputs(GameTime gameTime)
  {
    if (_currentBoard == null) return;

    var currentTime = gameTime.TotalGameTime;
    if (_pressedTime != null && currentTime - (TimeSpan)_pressedTime <
TimeSpan.FromMilliseconds(250)) return;
```

```
        foreach (var key in Keyboard.GetState().GetPressedKeys())
        {
            if (!_currentBoard.HandleKeyPressed(key)) continue;
            _pressedTime = currentTime;
            return;
        }

        if (_currentBoard.HandleGamepadButton(GamePad.GetState(PlayerIndex.One)))
        {
            _pressedTime = currentTime;
            return;
        }

        if (Mouse.GetState().LeftButton == ButtonState.Pressed &&
            _currentBoard.HandleMouseLeftButton(Mouse.GetState().X, Mouse.GetState().Y))
        {
            _pressedTime = currentTime;
        }
    }

    public virtual void Draw(SpriteBatch spriteBatch)
    {
        _boardSelectionPage?.Draw(spriteBatch);
        _currentBoard?.Draw(spriteBatch);
    }
}
```

TitleScene.cs –

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
```

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace PicturePuzzle;

public class TitleScene : IScene
{
    private TimeSpan? _pressedTime;
    private Game1 _game1;
    private Texture2D _logoTexture;
    private Button _startButton;
    private Button _exitButton;

    public TitleScene(Game1 game1)
    {
        _game1 = game1;
        int centerX = _game1.Window.ClientBounds.Width / 2;
        _startButton = new Button("sprites/common/start_button_unhovered",
"sprites/common/start_button_hovered",
            () => game1.CurrentScene = new GameplayScene(game1), game1);
        _startButton.X = centerX - _startButton.Width - 80;
        _startButton.Y = _game1.Window.ClientBounds.Height - _startButton.Height - 140;
        _exitButton = new Button("sprites/common/exit_button_unhovered",
"sprites/common/exit_button_hovered",
            () => game1.Exit(), game1);
        _exitButton.X = centerX + 80;
        _exitButton.Y = _game1.Window.ClientBounds.Height - _exitButton.Height - 140;
        LoadTextures(game1.Content);
    }

    private void LoadTextures(ContentManager content)
    {
```

```
        _logoTexture = content.Load<Texture2D>("sprites/title_scene/logo");
    }

    public virtual void Update(GameTime gameTime, GraphicsDeviceManager graphics)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
            Keyboard.GetState().IsKeyDown(Keys.Escape))
            _game1.Exit();

        if (Keyboard.GetState().IsKeyDown(Keys.Enter))
        {
            _game1.CurrentScene = new GameplayScene(_game1);
        }

        _startButton.Update();
        _exitButton.Update();
    }

    public virtual void Draw(SpriteBatch spriteBatch)
    {
        int xPos = _game1.Window.ClientBounds.Width / 2 - _logoTexture.Width / 2;
        int yPos = _game1.Window.ClientBounds.Height / 2 - _logoTexture.Height / 2 - 80;
        spriteBatch.Draw(_logoTexture, new Vector2(xPos, yPos), Color.White);
        _startButton.Draw(spriteBatch);
        _exitButton.Draw(spriteBatch);
    }
}
```

AbstractBoard.cs

```
using System;
using System.Collections.Generic;
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace PicturePuzzle;

public abstract class AbstractBoard
{
    private const int BoardTopLeftX = 10;
    private const int BoardTopLeftY = 10;

    private readonly List<Block> _blocks = new();
    private readonly Dictionary<string, Texture2D> _blockTextures = new();
    private readonly Dictionary<Texture2D, string> _blockTexturesReversed = new();
    private readonly List<string> _textureNamesInOrder;
    private Texture2D _boardBackgroundTexture = null!;
    private readonly SpriteFont _04BFont;
    private readonly Game1 _game1;

    private int _controlledBlockIndex;
    private TimeSpan? _startingTime;
    private readonly int _totalTimeInSeconds;
    private readonly int _gridSize;

    private int TotalBlocks => _gridSize * _gridSize;
    public string BoardName { get; }
    public bool HasEnded { get; private set; }
    public bool HasWon { get; private set; }
    public int Moves { get; private set; }
    public int TimeLeft { get; private set; }
```

```
    public AbstractBoard(Game1 game1, List<string> textureNamesInOrder, string
boardName, int totalTimeInSeconds, int gridSize = 3)
    {
        _game1 = game1;
        _gridSize = gridSize;
        _textureNamesInOrder = textureNamesInOrder;
        BoardName = boardName;
        TimeLeft = totalTimeInSeconds;
        _totalTimeInSeconds = totalTimeInSeconds;
        HasEnded = false;
        _04BFont = _game1.Content.Load<SpriteFont>("fonts/04B_30");
        LoadTextures();
        LoadBlocks();
    }

    private void LoadTextures()
    {
        foreach (var name in _textureNamesInOrder)
        {
            if (name == "null") continue;
            var texture2D = _game1.Content.Load<Texture2D>(name);
            _blockTextures.Add(name, texture2D);
            _blockTexturesReversed.Add(texture2D, name);
        }

        _boardBackgroundTexture =
_game1.Content.Load<Texture2D>("sprites/common/board_background");
    }

    private void LoadBlocks()
    {
```

```
        int blockStartX = BoardTopLeftX + 5;
        int blockStartY = BoardTopLeftY + 85;
        int blockWidth = 540 / _gridSize;
        int blockHeight = 540 / _gridSize;
        // ReSharper disable UselessBinaryOperation
        Block[,] blockArray = new Block[_gridSize,_gridSize];
        for (int i = 0; i < _gridSize; i++)
        {
            for (int j = 0; j < _gridSize; j++)
            {
                blockArray[i,j] = new Block(blockStartX + blockWidth * i, blockStartY +
blockHeight * j);
            }
        }

        for (int i = 0; i < _gridSize; i++)
        {
            for (int j = 0; j < _gridSize; j++)
            {
                int right = (j + 1 > _gridSize - 1) ? -1 : (j + 1);
                int left = (j - 1 < 0) ? -1 : (j - 1);
                int down = (i + 1 > _gridSize - 1) ? -1 : (i + 1);
                int up = (i - 1 < 0) ? -1 : (i - 1);

                blockArray[j, i].Up = (up == -1) ? null : blockArray[j, up];
                blockArray[j, i].Right = (right == -1) ? null : blockArray[right, i];
                blockArray[j, i].Down = (down == -1) ? null : blockArray[j, down];
                blockArray[j, i].Left = (left == -1) ? null : blockArray[left, i];

                _blocks.Add(blockArray[j, i]);
            }
        }
```

```
    List<string> randomisedTextures = RandomiseTextures();
    for (int i = 0; i < _gridSize * _gridSize; i++)
    {
        if (randomisedTextures[i] == "null")
        {
            _controlledBlockIndex = i;
            continue;
        }

        _blocks[i].Texture = _blockTextures[randomisedTextures[i]];
    }
}

private List<string> RandomiseTextures()
{
    List<string> randomTextures = new();
    Random random = new Random();
    List<int> acquiredIndexes = new();

    // Move "null" to first
    List<string> source = new() { "null" };
    foreach (var name in _textureNamesInOrder)
    {
        if (name != "null") source.Add(name);
    }

    for (int i = 0; i < TotalBlocks; i++)
    {
        int r = (int)random.NextInt64(0, TotalBlocks);
        while (acquiredIndexes.Contains(r))
```

```
        {
            r = (int)random.NextInt64(0, TotalBlocks);
        }

        randomTextures.Add(source[r]);
        acquiredIndexes.Add(r);
    }

    if (IsSolvable(acquiredIndexes))
        return randomTextures;
    return RandomiseTextures();
}

private bool IsSolvable(List<int> indexes)
{
    int inversions = 0;
    for (int i = 0; i < TotalBlocks - 1; i++)
    {
        for (int j = i + 1; j < TotalBlocks; j++)
        {
            if (indexes[j] > 0 && indexes[i] > 0 && indexes[i] > indexes[j])
                inversions++;
        }
    }

    return (inversions % 2 == 0);
}

public bool HandleKeyPressed(Keys keyboardKey)
{
    switch (keyboardKey)
```

```
    {
        case Keys.Up:
            HandleDownMovement();
            return true;
        case Keys.Right:
            HandleLeftMovement();
            return true;
        case Keys.Down:
            HandleUpMovement();
            return true;
        case Keys.Left:
            HandleRightMovement();
            return true;
        default:
            return false;
    }
}

public bool HandleMouseLeftButton(int mouseX, int mouseY)
{
    foreach (Block block in _blocks)
    {
        if (!block.Contains(mouseX, mouseY)) continue;

        if (block.Up != null && block.Up.Texture == null)
        {
            block.SwapTextures(block.Up);
            _controlledBlockIndex = _blocks.IndexOf(block);
            Moves++;
        }
        else if (block.Right != null && block.Right.Texture == null)
```

```
        {
            block.SwapTextures(block.Right);
            _controlledBlockIndex = _blocks.IndexOf(block);
            Moves++;
        }
        else if (block.Down != null && block.Down.Texture == null)
        {
            block.SwapTextures(block.Down);
            _controlledBlockIndex = _blocks.IndexOf(block);
            Moves++;
        }
        else if (block.Left != null && block.Left.Texture == null)
        {
            block.SwapTextures(block.Left);
            _controlledBlockIndex = _blocks.IndexOf(block);
            Moves++;
        }

        return true;
    }

    return false;
}

public bool HandleGamepadButton(GamePadState gamePadState)
{
    if (gamePadState.IsButtonDown(Buttons.DPadUp))
    {
        HandleDownMovement();
        return true;
    }
```

```
    if (gamePadState.IsButtonDown(Buttons.DPadRight))
    {
        HandleLeftMovement();
        return true;
    }
    if (gamePadState.IsButtonDown(Buttons.DPadDown))
    {
        HandleUpMovement();
        return true;
    }
    if (gamePadState.IsButtonDown(Buttons.DPadLeft))
    {
        HandleRightMovement();
        return true;
    }

    return false;
}

public void HandleUpMovement()
{
    Block controlledBlock = _blocks[_controlledBlockIndex];
    if (controlledBlock.Up != null)
    {
        controlledBlock.SwapTextures(controlledBlock.Up);
        _controlledBlockIndex = _blocks.IndexOf(controlledBlock.Up);
        Moves++;
    }
}
```

```
public void HandleRightMovement()
{
    Block controlledBlock = _blocks[_controlledBlockIndex];
    if (controlledBlock.Right != null)
    {
        controlledBlock.SwapTextures(controlledBlock.Right);
        _controlledBlockIndex = _blocks.IndexOf(controlledBlock.Right);
        Moves++;
    }
}

public void HandleDownMovement()
{
    Block controlledBlock = _blocks[_controlledBlockIndex];
    if (controlledBlock.Down != null)
    {
        controlledBlock.SwapTextures(controlledBlock.Down);
        _controlledBlockIndex = _blocks.IndexOf(controlledBlock.Down);
        Moves++;
    }
}

public void HandleLeftMovement()
{
    Block controlledBlock = _blocks[_controlledBlockIndex];
    if (controlledBlock.Left != null)
    {
        controlledBlock.SwapTextures(controlledBlock.Left);
        _controlledBlockIndex = _blocks.IndexOf(controlledBlock.Left);
        Moves++;
    }
```

```
    }

    public bool IsArranged()
    {
        for (var i = 0; i < TotalBlocks; i++)
        {
            if (_blocks[i].Texture == null)
            {
                if (_textureNamesInOrder[i] != "null")
                {
                    return false;
                }

                continue;
            }

            if (_blockTexturesReversed[_blocks[i].Texture!] != _textureNamesInOrder[i])
            {
                return false;
            }
        }

        return true;
    }

    public void Update(GameTime gameTime)
    {
        if (HasEnded) return;

        _startingTime ??= gameTime.TotalGameTime;
        TimeLeft = _totalTimeInSeconds - (int)(gameTime.TotalGameTime -
(TimeSpan)_startingTime).TotalSeconds;
```

```
        if (IsArranged())
        {
            HasWon = true;
            HasEnded = true;
            return;
        }

        if (TimeLeft <= 0)
        {
            HasWon = false;
            HasEnded = true;
        }
    }

    public void Draw(SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(_boardBackgroundTexture, new Vector2(BoardTopLeftX,
BoardTopLeftY), Color.White);
        _blocks.ForEach(block => block.Draw(spriteBatch));

        spriteBatch.DrawString(_04BFont, $"Moves: {Moves}", new Vector2(BoardTopLeftX +
15, BoardTopLeftY + 57),
            new Color(129, 23, 27), 0, Vector2.Zero, 0.8f, SpriteEffects.None, 0.5f);

        Vector2 textMiddlePoint = _04BFont.MeasureString(BoardName) / 2;
        // ReSharper disable once PossibleLossOfFraction
        spriteBatch.DrawString(_04BFont, BoardName,
            new Vector2(BoardTopLeftX + _boardBackgroundTexture.Width / 2,
                BoardTopLeftY + 40), new Color(84, 8, 4), 0, textMiddlePoint, 1.6f,
SpriteEffects.None, 0.5f);
```

```
        var x = BoardTopLeftX + _boardBackgroundTexture.Width -
_04BFont.MeasureString($"Time: {TimeLeft}").X + 12;
        spriteBatch.DrawString(_04BFont, $"Time: {TimeLeft}",
            new Vector2(x, BoardTopLeftY + 57), new Color(129, 23, 27), 0, Vector2.Zero, 0.8f,
SpriteEffects.None,
            0.5f);
    }
}
```

BatmanBoard.cs –

```
using System.Collections.Generic;


namespace PicturePuzzle;


public class BatmanBoard : AbstractBoard
{
    private static readonly List<string> TextureNamesInOrder = new()
    {
        "sprites/batman_board/batman_0_0",
        "sprites/batman_board/batman_0_1",
        "sprites/batman_board/batman_0_2",
        "sprites/batman_board/batman_0_3",
        "sprites/batman_board/batman_1_0",
        "sprites/batman_board/batman_1_1",
        "sprites/batman_board/batman_1_2",
        "sprites/batman_board/batman_1_3",
        "sprites/batman_board/batman_2_0",
        "sprites/batman_board/batman_2_1",
        "sprites/batman_board/batman_2_2",
        "sprites/batman_board/batman_2_3",
        "sprites/batman_board/batman_3_0",
        "sprites/batman_board/batman_3_1",
```

```
        "sprites/batman_board/batman_3_2",
        // "sprites/batman_board/batman_3_3",
        "null",
    };

    public BatmanBoard(Game1 game1, int totalTimeInSeconds)
        : base(game1, TextureNamesInOrder, "Batman Board", totalTimeInSeconds, gridSize: 4)
    { }
}
```

EmojiBoard.cs –

```
using System.Collections.Generic;

namespace PicturePuzzle;

public class EmojiBoard : AbstractBoard
{
    private static readonly List<string> TextureNamesInOrder = new()
    {
        "sprites/emoji_board/block_1",
        "sprites/emoji_board/block_2",
        "sprites/emoji_board/block_3",
        "sprites/emoji_board/block_4",
        "sprites/emoji_board/block_5",
        "sprites/emoji_board/block_6",
        "sprites/emoji_board/block_7",
        "sprites/emoji_board/block_8",
        "null",
    };

    public EmojiBoard(Game1 game1, int totalTimeInSeconds)
```

```
        : base(game1, TextureNamesInOrder, "Emoji Board", totalTimeInSeconds)
    { }
}
```

SimpleBoard.cs –

```
using System.Collections.Generic;

namespace PicturePuzzle;

public class SimpleBoard : AbstractBoard
{
    private static readonly List<string> TextureNamesInOrder = new()
    {
        "sprites/simple_board/block_1",
        "sprites/simple_board/block_2",
        "sprites/simple_board/block_3",
        "sprites/simple_board/block_4",
        "sprites/simple_board/block_5",
        "sprites/simple_board/block_6",
        "sprites/simple_board/block_7",
        "sprites/simple_board/block_8",
        "null",
    };

    public SimpleBoard(Game1 game1, int totalTimeInSeconds)
        : base(game1, TextureNamesInOrder, "Simple Board", totalTimeInSeconds)
    { }
}
```

Block.cs –

```
using Microsoft.Xna.Framework;
```

```
using Microsoft.Xna.Framework.Graphics;

namespace PicturePuzzle;

public class Block
{
    public Texture2D? Texture { get; set; }
    private readonly int _x;
    private readonly int _y;

    public Block? Up;
    public Block? Right;
    public Block? Down;
    public Block? Left;

    public Block(int x, int y)
    {
        _x = x;
        _y = y;
    }

    public void Draw(SpriteBatch spriteBatch)
    {
        if (Texture != null)
        {
            spriteBatch.Draw(Texture, new Vector2(_x + 5, _y + 5), Color.White);
        }
    }

    public bool Contains(int x, int y) => Texture != null && x >= _x && x <= _x +
Texture.Width && y >= _y && y <= _y + Texture.Height;
```

```
    public void SwapTextures(Block fromBlock)
    {
        Texture2D temp = Texture;
        Texture = fromBlock.Texture;
        fromBlock.Texture = temp;
    }
}
```

Button.cs –

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace PicturePuzzle;

public class Button
{
    private Texture2D _unHoveredTexture;
    private Texture2D _hoveredTexture;

    private Action _onClick;
    private Game1 _game1;

    private bool _hovered;

    public int X { get; set; }
    public int Y { get; set; }
    public int Width { get; private set; }
    public int Height { get; private set; }
    public bool Hovered => _hovered;
```

```
    public Button(string unHoveredTextureName, string hoveredTextureName, Action onClick,
Game1 game1)
    {
        _game1 = game1;
        _unHoveredTexture = _game1.Content.Load<Texture2D>(unHoveredTextureName);
        _hoveredTexture = _game1.Content.Load<Texture2D>(hoveredTextureName);
        _onClick = onClick;
        X = 0;
        Y = 0;
        Width = _unHoveredTexture.Width;
        Height = _unHoveredTexture.Height;
    }

    public void Update()
    {
        int mouseX = Mouse.GetState().X;
        int mouseY = Mouse.GetState().Y;

        if (mouseX >= X && mouseX <= X + _hoveredTexture.Width && mouseY >= Y &&
mouseY <= Y + _hoveredTexture.Height)
        {
            _hovered = true;
            if (Mouse.GetState().LeftButton == ButtonState.Pressed)
                _onClick.Invoke();
        }
        else
        {
            _hovered = false;
        }
    }
```

```
    public void Draw(SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(_hovered ? _hoveredTexture : _unHoveredTexture, new Vector2(X, Y), Color.White);
    }
}
```