Nine Men’s Morris

Technical documentation

Emilia Haramina

Table of Contents

Nine Men’s Morris game description

The game implemented as part of this task is the Nine Men’s Morris game in a 2D setting. It was implemented using Unity version 2023.2.20f1 for Windows.

The game consists of three scenes: a main menu, a loading screen, and the actual game.

The main menu of the game allows players to start the game and change settings for players, for sound, and for the game options.

The loading screen does not contain any interactable elements but is used as a transition when loading scenes. It shows the percentage of the loading progress of the next scene.

The actual game consists of three phases: placing pieces, moving pieces and “flying”. Players take turns. In the first part of the game, the placing phase, players place pieces on the board. When a new mill is formed, which consists of three pieces from one player in a straight line, the player who formed a mill can remove one of the opposing player’s pieces from the board. Once players have placed down all their pieces, the game moves on to the next phase, the moving phase. In this phase, players can move their pieces to neighbouring spots on the board, and mills can be formed as well, allowing players to, again, remove the other player’s pieces. Once a player has been left with only three pieces on the board, they enter the “flying” phase, which allows them to move their pieces anywhere on the board, instead of only their neighbouring spots. Finally, once a player has been left with only two pieces, meaning they are unable to form a mill, or cannot move their pieces, they lose the game and the game ends.

During the game, players can open a pause menu in which they can adjust the music or continue playing the game. Additionally, both in the pause menu and the victory menu, players can choose to restart the game or return to the main menu scene.

Implementation architecture

Game Logic

The **Point** class defines a point on the board. It contains information about its position, as well as animators for placing and removing a piece for that point and showing legal moves on the board. It also contains a player id to show which player currently has a piece on a **Point**. The position of a **Point** is defined by 3 coordinates, one for the circle index, one for the column index, and one for the row index. A **Point** can be made pickable or illegal. It also has a small circular symbol that shows when mills are shown on the board, and that **Point** is a part of a newly made mill, and an outline that is enabled when the piece on that **Point** is about to move.

The **Board** class contains a list of **Points** and defines the board players play the game on. When it is initialized, it instantiates **Points** and places **Points** and lines on the scene according to the given number of rings. When required, it updates the current state of the board depending on what player is currently playing and what phase the game is currently in. It does so by setting which **Points** are legal to click, and which are illegal. It can also clear the entire **Board** of legal moves and button listeners. Furthermore, it contains helper functions for finding neighbours of points, checking whether two points are neighbours on the **Board**, finding the number of completed mills with the given **Point** and managing **Point** animations and variables.

The **GameManager** class changes the phases of the game accordingly. It also defines the number of rings and pieces in the game. It manages the **Board** and allows the player to take an action depending on the phase of the game. The class also keeps track of whether the game is paused and does not allow the player to complete an action if the game is paused. It also manages which player's turn it is. It checks for win and lose conditions and ends the game.

Game information

The **GameInformationController** class changes UI text in the game showing the player informational messages about the state of the game, such as the number of pieces left for both players, the current instructions, and the victory menu.

General information

The **DefaultValues** class contains static variables that define the default values of many parts of the game. That includes whether the background music and sound effects are turned on, as well as their volume, the player ids, names, and color hex values, the values for borders around color choices, the player id for a free point on the board, the thickness of a line on the board, the outline scale of a point on the board, the number of ring and pieces, and the minimum number of ring and pieces.

The **PlayerPrefsKeys** class contains the keys for various player preferences, such as whether the background music and sound effects are playing, as well as their volume, the player names, and the player colors. It also initializes these values in the player preferences if they have not been initialized yet.

Player

The **PlayerManager** class manages information about both players. This includes their ids, names, colors, and the name of their name and color keys for the player preferences. It contains functions that check whether a name or color is available, and functions that change a player's name or color.

The **ColorChoice** class contains information about one of the player's color choices. It also makes these choices available, unavailable, or picked, depending on what both users had chosen as their colors.

Sound

The **MusicVolumeController** and the **SoundEffectsVolumeController** classes both rely on a scrollbar to disable or enable the volume of the background music or the sound effect, respectively.

The **MusicController** and the **SoundEffectsController** classes both manage whether the background music or sound effects should be playing, respectively, and their volumes. However, their functions differ, as the background music is constantly playing, while sound effects are only played after a specific action. Because of this, the **MusicController** class contains an **AudioSource** variable that plays the background music, while the **SoundEffectsController** relies on **AudioSources** passed as arguments and plays them only if sound effects should be playing.

The **SoundManager** class is used by classes managing UI to set whether the background music or sound effects are playing and changes the speaker icon depending on whether they are currently muted or not.

The **SoundEffectSource** class defines an **AudioSource** and uses the **SoundEffectsController** function to play a sound effect.

Menus

The **MenuManager** class manages menu functions that are the same for all menus, which are sound functions. All menus can toggle both background music and sound effects, as well as manually set whether they are playing or not.

The **MainMenuManager** class, in addition to the **MenuManager** class functions, manages several other functions available in the main menu and the settings menu. It allows players to toggle between the main menu and the settings menu. It also allows players to change their names, while alerting them if their name is invalid (empty or if it is the name of the opposing player), and their colors, not allowing them to pick a color the opponent has already picked. Additionally, it allows players to change the number of rings in the game and the number of pieces players start with, also alerting the players if they are invalid (too small, too large, or not a number). Finally, it allows players to exit the game.

The **PauseMenuManager** class, along with the functionality of the **MenuManager** class, checks whether a player has paused or unpaused the game with the *Escape* button, opening or closing the pause menu.

For both the main menu and the pause menu, buttons that load a scene, such as *Play* on the main menu and *Restart* and *Return to main menu* on the pause menu, the buttons were assigned functions for scene loading in the editor, so these functionalities are not implemented in the menu classes.

The **SettingsManager** class initializes settings that are available in both the main menu and the pause menu, which is the background music and the sound effects options.

Loading screen

One of the three scenes present in the implementation is the **LoadingScreen**. It is used as a transition between the loading of two scenes. It contains text showing the loading percentage of the scene that is loading.

The **SceneLoader** class is responsible for loading the next scene and unloading the previous scene. It also contains parameters for faded scene loading, so the loading screen would not flash by too quickly, and changes the loading screen text.

The **MenuSceneLoader** class is used to load a scene after a certain, defined, time passes.

The **SceneLoadTrigger** class contains functions that start scene loading with or without a loading screen.

The **SceneLoadingData** class contains static variables that define the names of the scene to load, of the scene to unload, and of the loading scene.

Playing instructions

The player starts the game in the main menu, seen in Fig. 1. In it, the player can choose to start the game by clicking on the *Play* button, open the settings using the *Settings* button, or exit the game by clicking the *Exit* button.

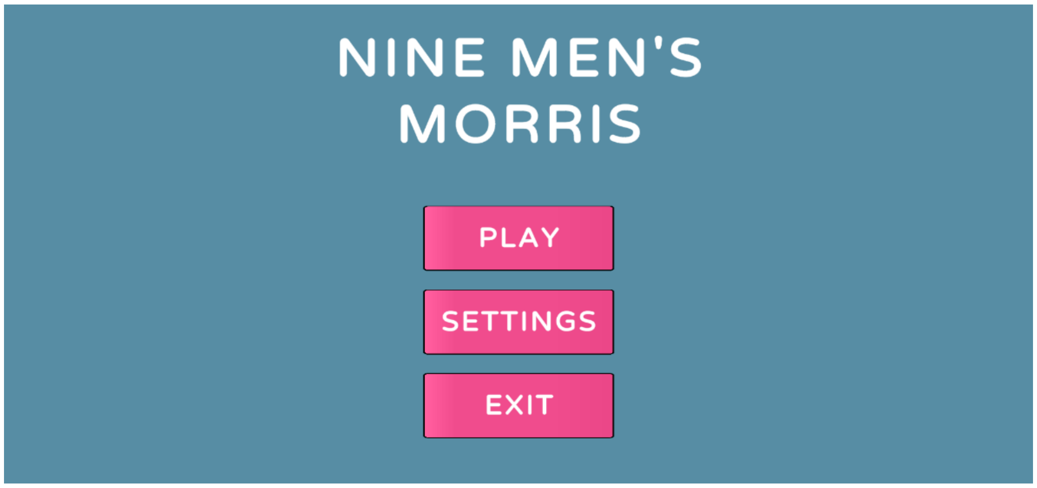

Figure 1 The main menu

Opening the settings results in the screen seen in Fig. 2. In this menu, players can choose their names and colors, as well as toggle and change the volume of the background music and the sound effects. They can also change the number of rings and pieces in the game. Finally, they can return to the main menu. The game does not allow the player to change the volume of the background music or the sound effects if they are turned off, respectively. The game also does not allow the player to pick a color that was picked by the other player. Additionally, the game warns the player is they have set a name that is empty or taken by the other player or if the number of rings of pieces is out of bounds. These values get reset once the played closes the settings menu, and the valid values can be seen once they open it again. The background music and sound effects settings, as well as the names and chosen colors of the players, are saved for the next session.

Once the player starts the game, they are met with a question, as seen in Fig. 3. The player that is chosen is the one that places their piece first in the game. The number of pieces in each player's hand is shown on the sides of the screen. Afterward, players are instructed to place a piece, as can be seen in Fig. 4. Legal moves in this phase are those that have no pieces on them. All legal moves are animated as a pulsating green circle. When the player plays a move, the other player is instructed to do the same. This repeats until a player forms a mill.

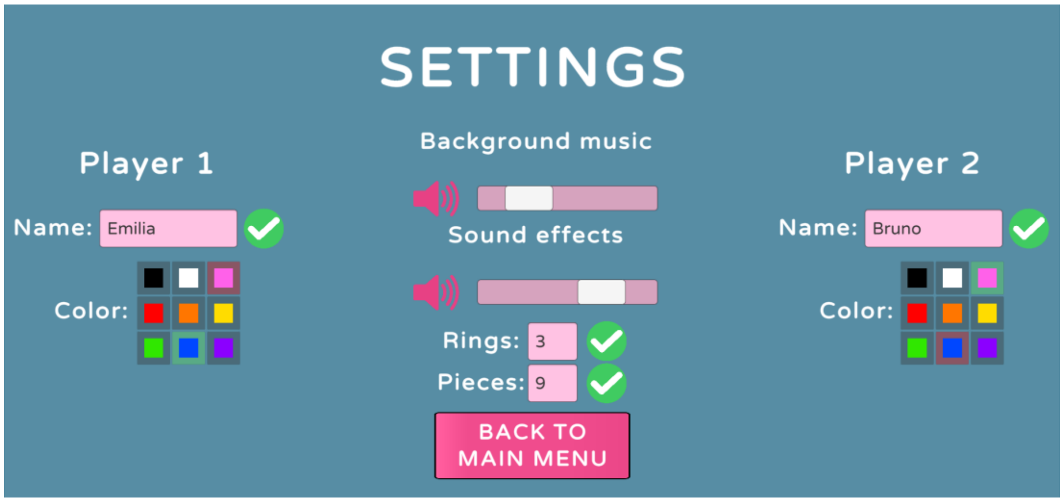

Figure 2 The settings menu

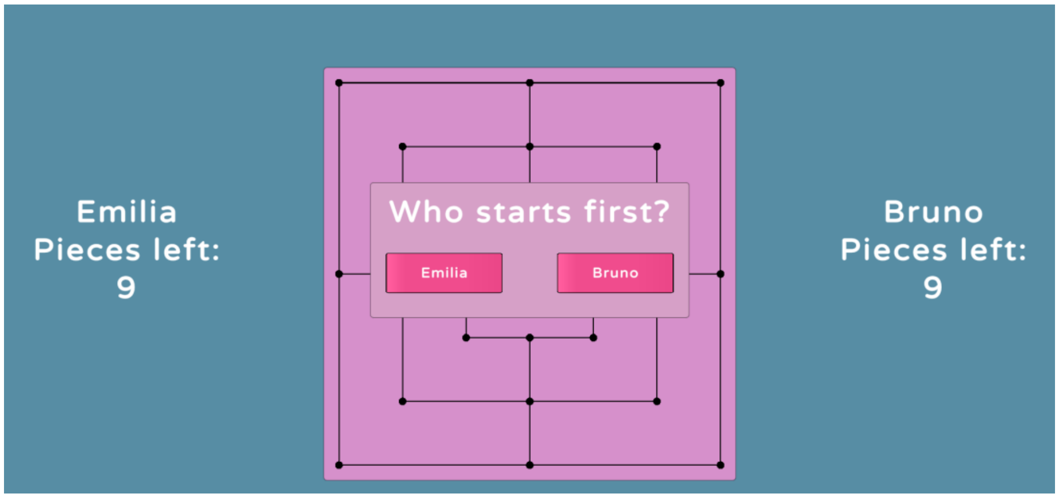

Figure 3 Game start question

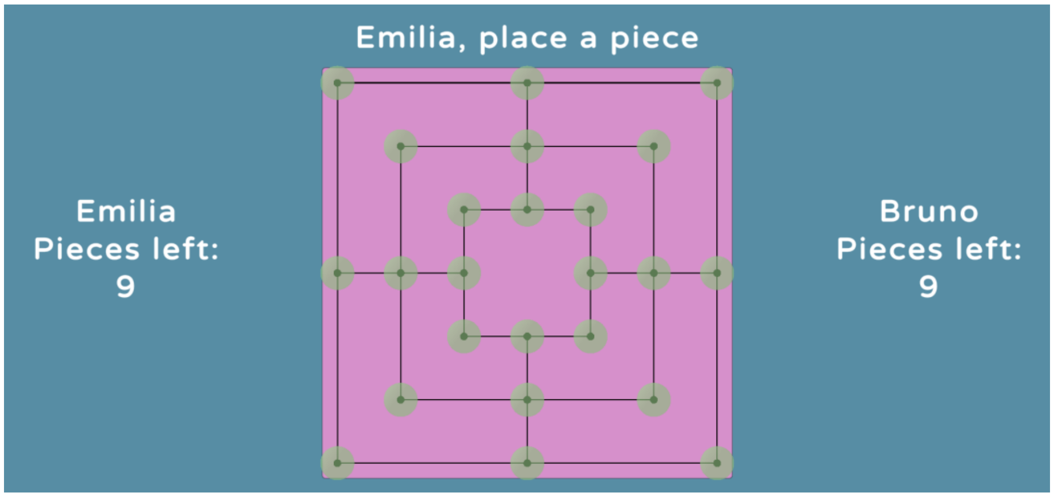

Figure 4 Placing a piece

A mill is formed when a player has placed three of their pieces in a line. When a mill is formed, the pieces in a newly created mill are highlighted with a small circle in their center. For each mill formed, the player is allowed to remove a piece from the other player, as seen in Fig. 5. However, the player is not allowed to remove a piece that is a part of a mill unless all the opposing player's pieces form a mill.

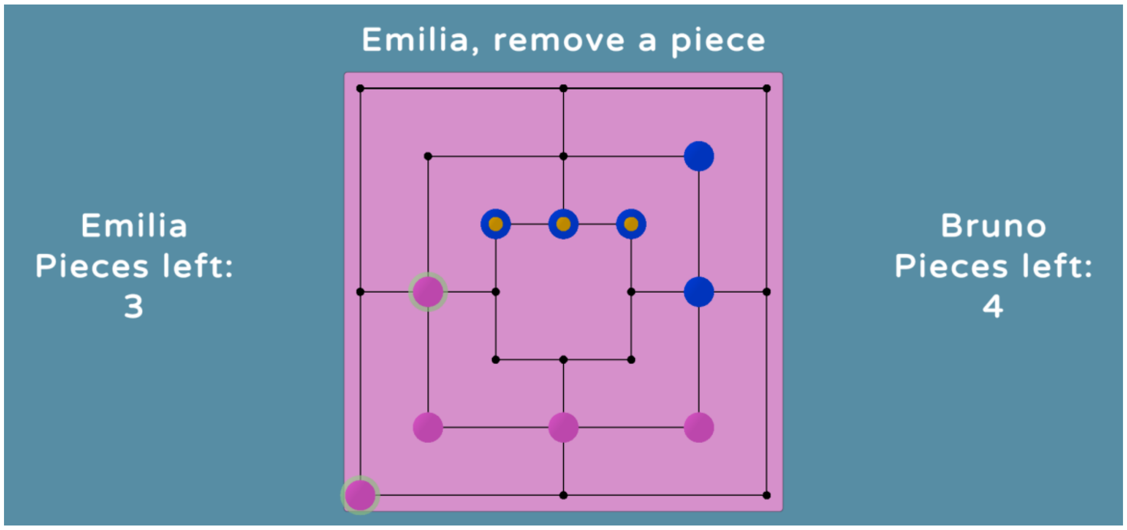

Figure 5 Removing a piece

Once both players have run out of pieces in their hands, players can move their pieces around the board to a neighbouring point, as seen in Fig. 6 and 7. Legal moves in this phase are only those pieces that have a neighbouring point that does not have a player's piece on it. If a player forms a mill by moving their pieces, they are again instructed to remove a piece belonging to the opposing player.

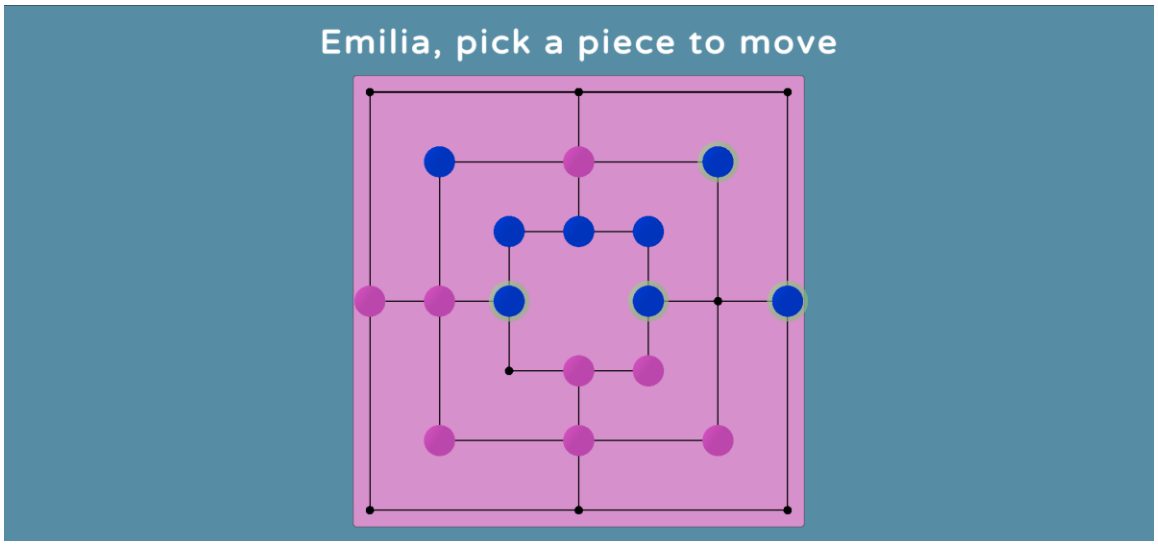

Figure 6 Choosing a piece to move

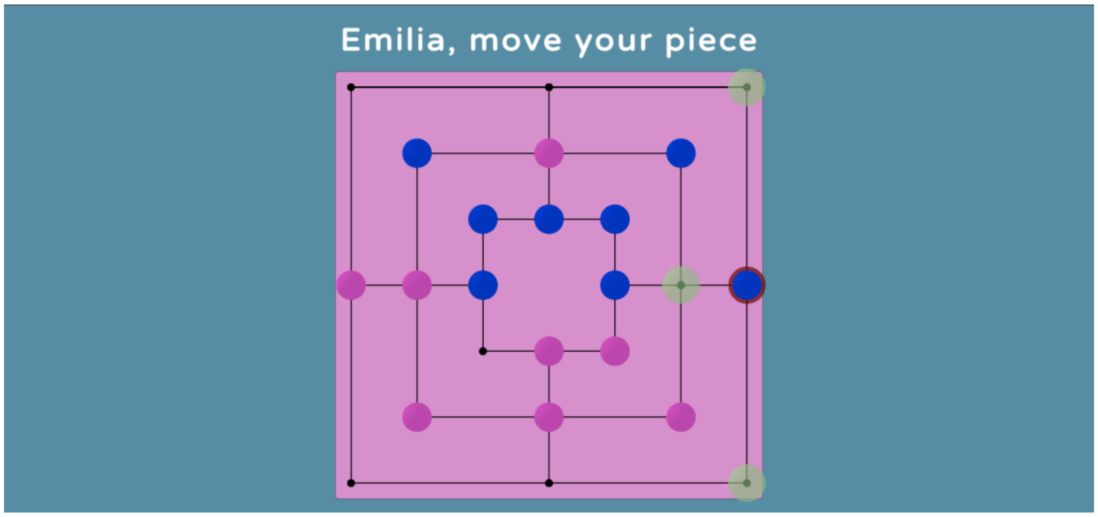

Figure 7 Moving a piece

Once a player has only 3 pieces left on the board, instead of moving only to a neighbouring point, they can move their pieces anywhere on the board, an action called flying, as seen in Fig. 8.

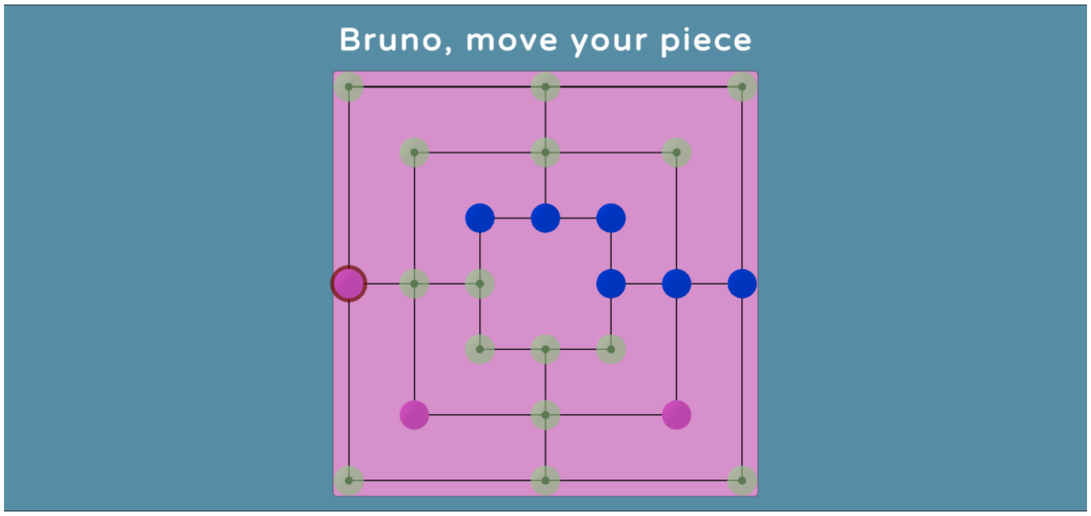

Figure 8 Flying

When a player is either left with only two pieces or they cannot move anywhere, they lose the game and a victory menu pops up, as seen in Fig. 9. Using this menu, players can restart the game using the *Restart* button or go back to the main menu using the *Return to main menu* button.

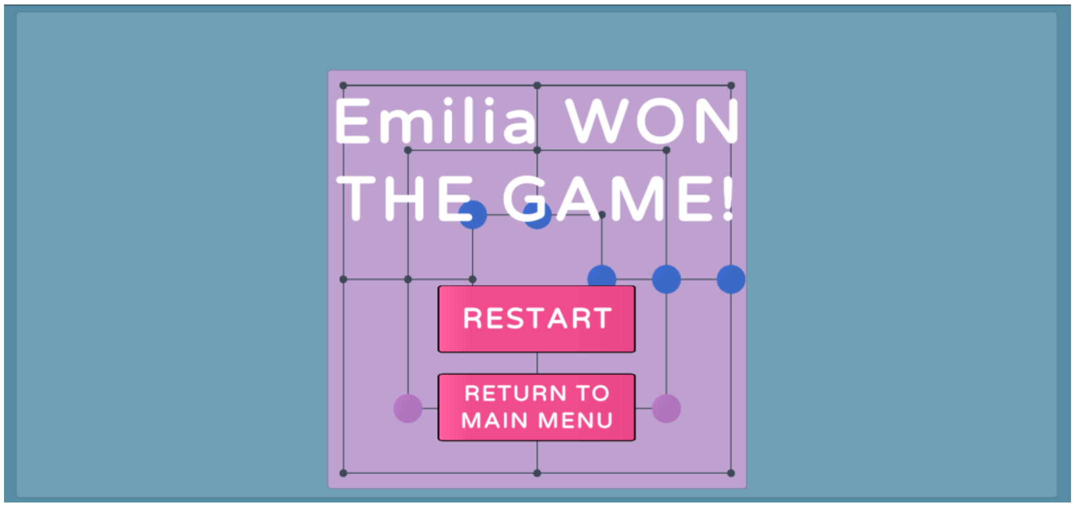

Figure 9 The victory menu

Additionally, if the player presses the *Escape* button while playing the game, they open the pause menu, seen in Fig. 10. In the pause menu, players can toggle the background music or sound effects and adjust their volume, continue the game either by clicking the *Escape* button again or clicking the *Continue* button, restart the game by using the *Return* button, or go back to the main menu using the *Return to main menu* button.

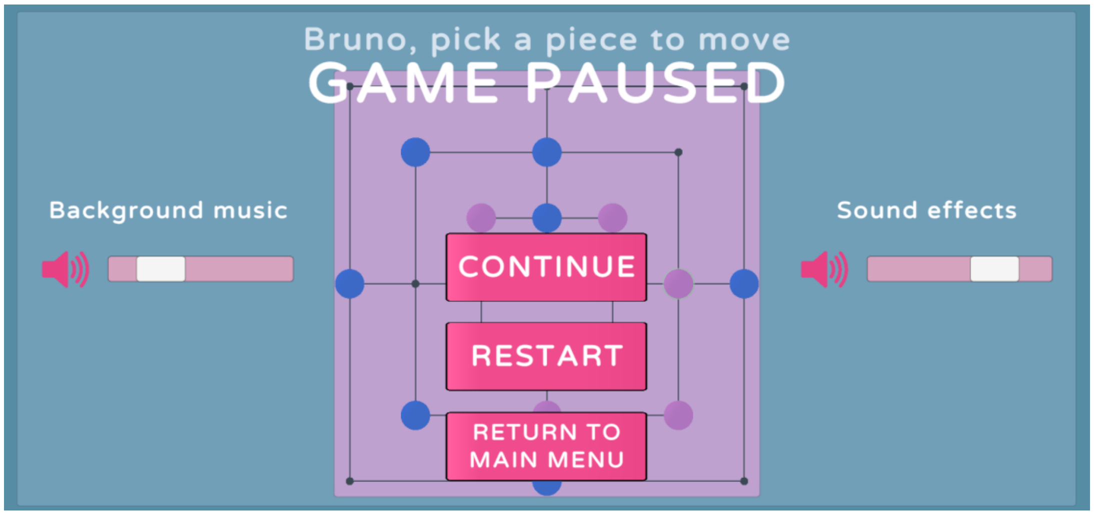

Figure 10 The pause menu