VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Samu Välimäki

MATCH-THREE GAME FOR WINDOWS

Information Technology
2019

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Samu Välimäki
Title	Match-three Game for Windows
Year	2019
Language	English
Pages	71
Name of Supervisor	Timo Kankaanpää

Video game development is a constantly growing market and casual games make a big part of the market. The aim of the thesis was to study different game engines to develop two prototypes, and to develop a game implementing the popular match-three genre of puzzle games in one engine.

The Unity and Clickteam Fusion game engines were studied for the case, and Fusion was chosen as the final platform.

The result is a fun and fast-paced puzzle game. A lot was learned from what goes to game development, such as UI and graphics design and how to control the events of the game to achieve good performance and satisfying gameplay.

Avainsanat Video game, match-three, Clickteam Fusion

CONTENTS

LIST OF FIGURES AND TABLES

1 INTRODUCTION

In this project a match-three game for the Microsoft Windows operating system was created. A match-three game is a two-dimensional puzzle video game where the player is given a board of easily distinguishable objects, and the player must swap the positions of two adjacent pieces in order to create matches of three or more objects and clear the board, with the game generating new pieces until the game is over.

Multiple game engines were considered for the project. Before the start of the production of the final product, two engines out of these were chosen and prototypes were developed for them in order to get familiar with them and inspect the different procedures that can be utilized to create a grid-based game. After this, one of the prototypes was chosen to be developed further to meet the requirements set for the project.

During the development cycle, multiple areas of game development were explored to ensure a complete experience, including the creation of interfaces, menus, graphics and sounds, alongside with what goes into creating satisfying and correctly paced gameplay.

2 REQUIREMENTS

The project was chosen certain requirements to evaluate its success. The features were divided into three categories: Must have-features, should have-features nice to have (optional) features. These features are shown in tables 1-3.

2.1 Must have-features

Table 1. List of must have-features.

Different menus	There needs to be proper menus to enter the game modes and activate other choices.
Playfield and objects	The regular gameplay is to take place on an 8x8 sized grid-based play area, each tile filled with an object (a piece). There must be five or six different object groups, and each generated piece belongs to one of these groups.
Swapping pieces around	The core gameplay consists of the player selecting a piece with the mouse and swapping it with another piece that is to the side, above or below the chosen piece. The chosen piece must be somehow highlighted for clarity. If the selected move is legal, as in forming a match, the positions of the pieces are swapped, and a match will occur. All other kinds of moves are illegal, and the game must recognize this and not allow such moves.
Acting out the matches	When a match is formed, the game must recognize this and destroy those pieces involved in a match. Multiple matches can be formed out of a single move. When the pieces are destroyed, the pieces above them must fall downwards and new ones to be generated to always have the board full of pieces. The pieces falling can cause other matches, and the control should not be given back to the player until the board has stabilized itself.
Changing the decision of the chosen piece	The player might conclude that they do not want to move the piece they have chosen. If a player clicks the piece they have selected, it will be deselected. Also, if a player chooses a piece, and afterwards chooses another piece that is out of range for swapping with the first piece, the selection will be transferred to that piece instead.
Scoring system	The game needs to have a scoring system to track the performance of the player. If this is not done, playing the game is unsatisfactory, as there is no reward for the actions of the player.
Keeping track of the game state	The game needs to keep track of the possible moves the player can make in each situation, and if there is a situation where there are no possible moves to be made, the game must recognize this and issue a “game over” to prevent itself from locking down.

2.2 Should have-features

Table 2. List of should have-features.

Multiple game modes	The game needs to have **at least** two separated modes of play: Timed and Standard. In timed mode, there is a timer, and the game ends when the timer reaches zero. The allowed play time can be a chosen parameter, for example between 1, 5 and 10 minutes. Standard mode ends only if there are no more moves available on the board. There can also be more game modes which can be planned later in the production.
State saving and restart	In normal mode, you should be able to exit the game at any point and save the state for later continuation, and it should start the game from the same position the next time the player starts playing. In timed mode, there should be a restart button, so the player can start over if the attempt is not going well.
Local leaderboards	To make the scoring system more meaningful, the best scores achieved by the player for each mode should be saved locally and be able to be viewed from the main menu.
Satisfactory gameplay	To make the game fun to play, the audiovisual design and responsiveness of the game must make the player feel good when they complete matches. This is subjective but should be strived for regardless.

2.3 Nice to have-features

Table 3. List of nice to have-features

Global leaderboards	In addition to local leaderboards, the game could have the capability to upload the scores achieved by players into an SQL database (or such) and the content of the database could be viewed in-game. In addition, there should be made an administrative page for the leaderboards so clearly cheated scores can be removed easily.
Twitch integration	As an experimental feature, the game could be controlled by connecting it to a Twitch.tv chat. This would have the game reading the text from the chosen chat and recognizing certain messages as gameplay commands and play out the game through those commands.
Steam features	If the game is ready otherwise and can be ready for release, using Steam’s features such as Steam Cloud or Steam Achievements can be investigated.
Android version	If the Windows version is satisfactory, the game can also be ported over to Android to increase playerbase.

3 OPTIONS FOR GAME ENGINES

There are numerous game engines that have been developed for the purpose of aiding game developers in their jobs. These engines handle the graphics rendering, sounds, physics and multiple other subjects of the games, and have lots of available information and community support. For this match-three game project, prototypes were developed with Unity and Clickteam Fusion. In addition to these engines, a couple other ones were also considered. In this part these game engines are shortly introduced.

3.1 Unity

Unity is an engine released in 2005 by Unity Technologies. Its primary purpose within game development is usually to be a 3D engine, but 2D development is also very active within its community. Unity is extremely versatile and allows development for dozens of different platforms for free, such as Windows, iOS, Android, Linux, PlayStation 4 and WebGL, just to name a few. This makes it a very good choice if the release for multiple platforms is high priority. The scripts for Unity are created in the C# language. It is an extremely popular engine and I also had previous experience on using it, so it was chosen to be the first prototype engine.

Popular games created with Unity: Hearthstone, Ori and the Blind Forest, Hollow Knight [1]

3.2 Clickteam Fusion

Clickteam Fusion is a 2D development software by Clickteam SARL. The company has been developing 2D engines since their establishment in 1993. As of 2019, the newest version of the software is Fusion 2.5. The engine can also be used to develop other kinds of software and multimedia, such as slideshows and general Windows applications.

Clickteam Fusion does not use a traditional scripting system with a text-based programming language, instead offering a graphical event system that is designed to allow the user to do pretty much anything, as well as offering a wide variety of plugins to add functionality. With the regular version of Fusion, only Windows applications are supported. iOS/Android/HTML5 platforms are also supported, but with additional costs. This is not the best case scenario but as I was primarily only looking to develop a Windows game, I could let it slide.

Fusion is a paid software, but I had received it at an earlier time on Steam, so I wanted to put it to use. The “visual” programming style sounded intriguing, as I had no previous experience of anything of the sort. I also figured that as a platform dedicated for 2D development, for my 2D game project it could be a very viable candidate. For these reasons, I chose it to be the second prototype engine.

Popular games created with Fusion: Freedom Planet, Five Nights at Freddy’s [2-4]

3.3 GameMaker Studio

GameMaker Studio is developed by YoYo Games and was first released in 1999. It is very similar to Fusion in a lot of aspects, such as including a visual programming language utilizing drag and drop mechanisms, but it also has its own text-based scripting language.

GMS is typically considered to be more of a beginner-oriented development software, but there are many good examples of what is possible to be done with it. I was mildly interested in picking it but considering I did not own it (the software is not free) I did not end up picking it. As I read about it, I also found many comments on its instability, something I did not find for Fusion or Unity. However, I am still interested in trying development with GMS in the future. It does support multiple platforms out of the box, with Nintendo Switch being the most recent addition.

Popular games created with GameMaker Studio: Undertale, Hotline Miami, Nuclear Throne [5]

3.4 Construct

Construct is a 2D game editor, first released in 2007 by Scirra, being the newest engine on the list. The current version Construct 2 was released in 2011, with Construct 3 in development at the moment. Construct seems to be a very powerful and interesting engine, utilizing a lot of premade features which could produce impressive results with seemingly little work. Construct is available for free, but there is a paid version available for professionals if they desire to sell their work.

I am definitely very interested in familiarizing myself with Construct but seeing as the engine is kind of obscure and did not seem to have that much community support behind it, I felt that it was not the strongest option. The scripting system seems quite similar to Fusion, but Fusion has been around for way longer and has been able to build a strong community and plugin library, but Construct is still growing and keeps getting more interesting. The platform support is a bit more limited than in the other programs, with Windows, Android and iOS being the primary platforms.

Popular games created with Construct: Our Darker Purpose [6], [7]

4 CREATION OF PROTOTYPES

Before working on the final version of the game, the goal in the project was to develop two prototypes on two different engines. The two game engines chosen ended up being Clickteam Fusion and Unity.

Clickteam Fusion was chosen despite being a paid software, as it seemed like a powerful and interesting engine to attempt developing a 2D game in. The engine not being free was not an issue, as I had purchased it on Steam a while back. There was going to be a lot of studying to do for development, but a part of the interest was for the matter of finding out if the visual scripting style would make development easier or harder, and in which ways.

Unity was chosen as the second engine for being a widely used and versatile engine, with which I have also had previous experience, working on simple games such as a sidescrolling platformer and a space shooter. The primary interest was in finding out how one would go about creating a grid-based puzzle game inside of Unity.

4.1 Prototype details

These prototypes will not fit all initial requirements, but most of them. The act of reading the number of possible moves and assigning a game over was chosen to be too advanced for a gameplay prototype, but all other gameplay related functions should work as intended. Menus are also not necessary.

This concludes that the prototype should:

- Generate the 8x8 playing field without having matches happen in the start of the game
- Allow for changing the positions of two adjacent objects only if it results in a match being created
- Have a basic scoring system in place
- Have matches function properly without graphical glitches and have new pieces fall from above as intended.

4.2 Unity prototype

Because of the previous experience with the Unity engine, no basic tutorials were needed. However, having only done physics-based games before on the engine, there was no clear starting point to how to go about creating a 2D puzzle game on the platform. This was found through a Unity tutorial by Jeff Fisher [8]. This tutorial included a download with it with a lot of premade assets, but it would have to be edited to fit the requirements of this project.

The article explains most of the code used and repeating it would take a very long time, so this part will mainly focus on what was changed to make the game fit the needs of this project. An introduction to the actual game engine will also not be provided, as Unity is very common in game development nowadays and it uses the common C# language for scripting.

4.2.1 List of Unity GameObjects

The default building block in Unity is called a GameObject and these are placed in a “scene” such as “Game” containing the gameplay elements. These objects utilize the C# scripts that can be assigned to them to create the gameplay. These are the GameObjects used in this prototype.

Table 4. List of Unity GameObjects

Main Camera	The default camera in Unity. Stationary in this project
GameManager	Handles things such as scene selection
GUIManagerCanvas	Contains the ScoreTxt object
ScoreTxt	Used to display the score of the player
SFXManager	Handles the sound effects
BoardManager	Handles the creation of the game board and manages finding null tiles and shifting
Tile	The basic gameplay element that is used to fill the game board

Figure 1 shows the completed Unity prototype.

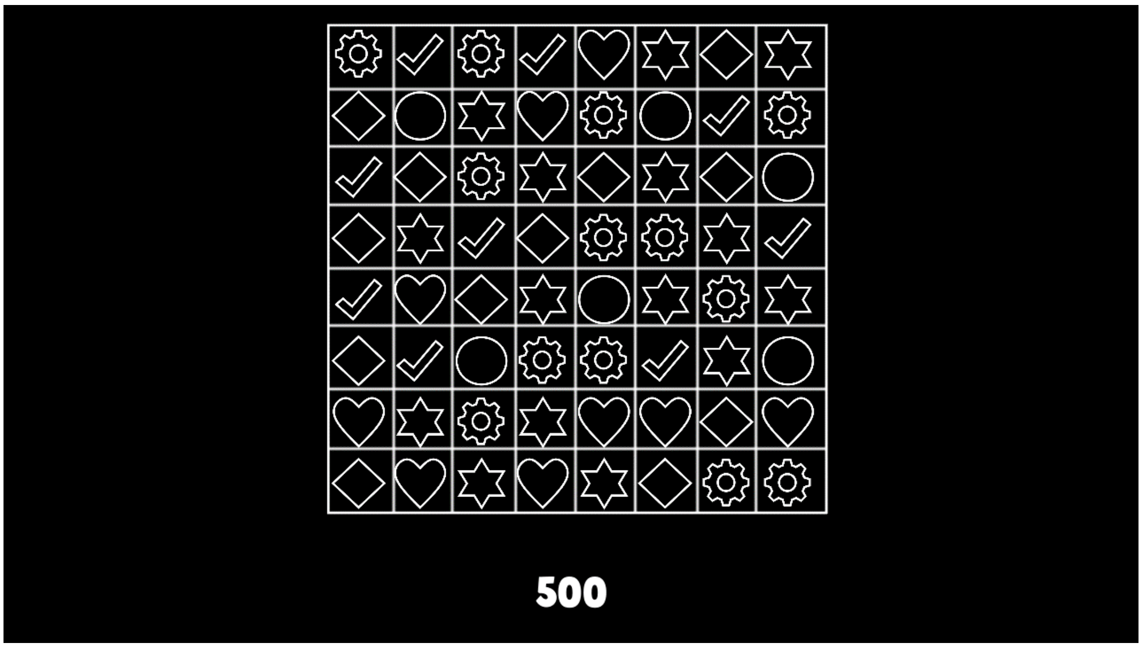
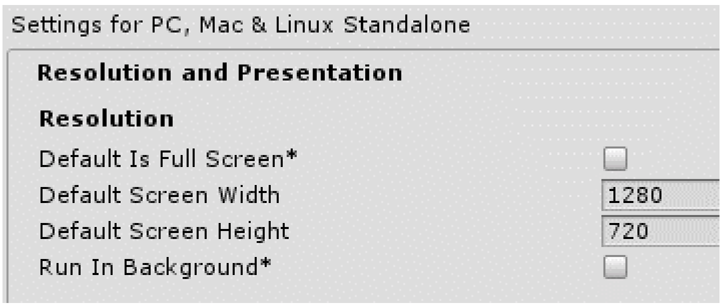


Figure 1. The completed Unity prototype.

4.2.2 Changes to the tutorial project

The original tutorial presented a game designed for mobile phones. The purpose of the project was to create a game for Windows, so the resolution had to be changed to be fitting. This was done by changing the value in Unity's PlayerSettings as shown in Figure 2.



Settings for PC, Mac & Linux Standalone

Resolution and Presentation

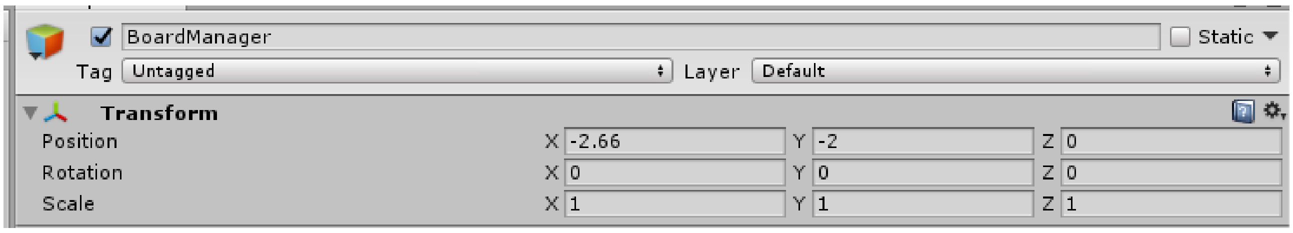
Resolution

Default Is Full Screen*	☐
Default Screen Width	1280
Default Screen Height	720
Run In Background*	☐

Figure 2. Resolution in Unity.

This resulted in the game always running in a 1280x720 window. The aspect ratio in the "Game" display also had to be changed to 16:9 for testing to function properly.

The position of the BoardManager GameObject decides where the Tile objects will be created. Figure 3 shows the change to fit the new screen size.



BoardManager

Static

Tag Untagged

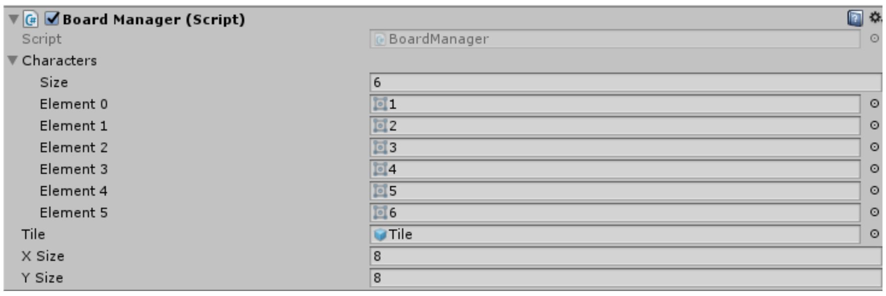
Layer Default

Transform

Position	X	-2.66	Y	-2	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Figure 3. Position of BoardManager.

New graphics were created for the Tile objects, and the size of the game board was changed to 8x8 in BoardManager. (Figure 4).

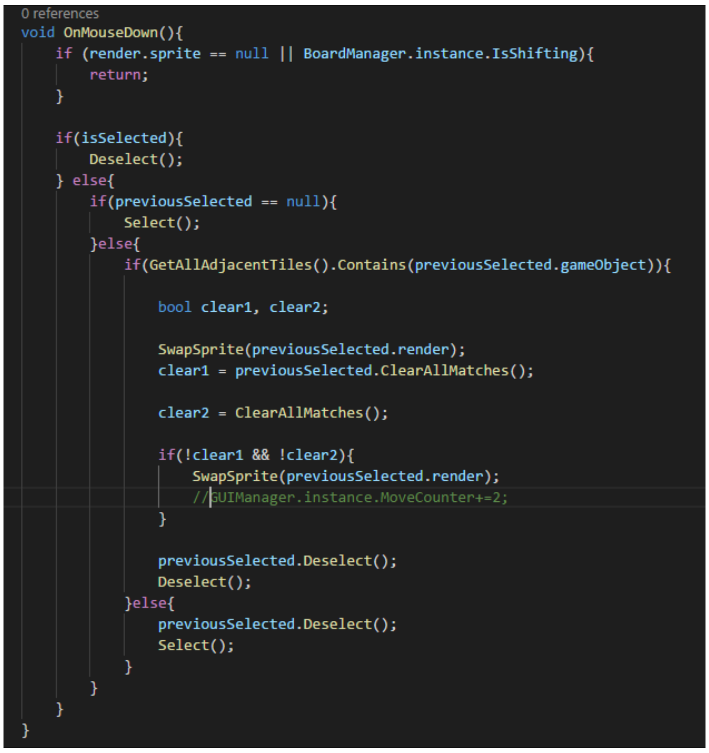


Board Manager (Script)

Script	BoardManager
Characters	
Size	6
Element 0	1
Element 1	2
Element 2	3
Element 3	4
Element 4	5
Element 5	6
Tile	Tile
X Size	8
Y Size	8

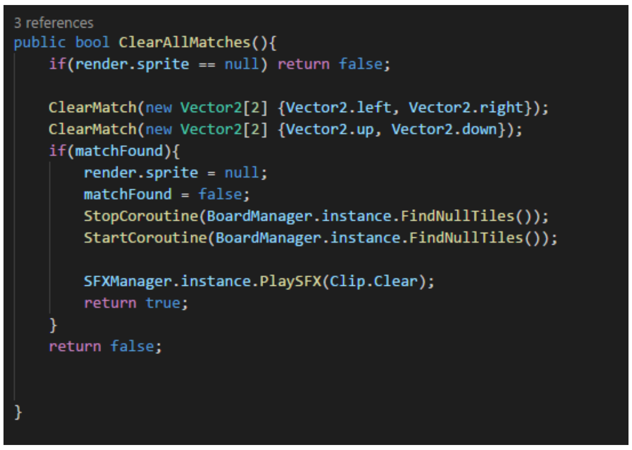
Figure 4. Editing the BoardManager.

The tutorial project had a “move counter” and allowed moves that did not result in matches. This is not correct for the kind of game that is being made, so the move counter was completely removed, and the code was reworked to only allow swaps that result in matches. This is achieved through making the ClearAllMatches method return a Boolean value, if no matches were achieved, the swap would be redone, canceling the swap. Figures 5 and 6 display these changes.



```
0 references
void OnMouseDown(){
    if (render.sprite == null || BoardManager.instance.IsShifting){
        return;
    }

    if(isSelected){
        Deselect();
    } else{
        if(previousSelected == null){
            Select();
        }else{
            if(GetAllAdjacentTiles().Contains(previousSelected.gameObject)){

                bool clear1, clear2;

                SwapSprite(previousSelected.render);
                clear1 = previousSelected.ClearAllMatches();

                clear2 = ClearAllMatches();

                if(!clear1 && !clear2){
                    SwapSprite(previousSelected.render);
                    //GUIManager.instance.MoveCounter+=2;
                }

                previousSelected.Deselect();
                Deselect();
            }else{
                previousSelected.Deselect();
                Select();
            }
        }
    }
}
```

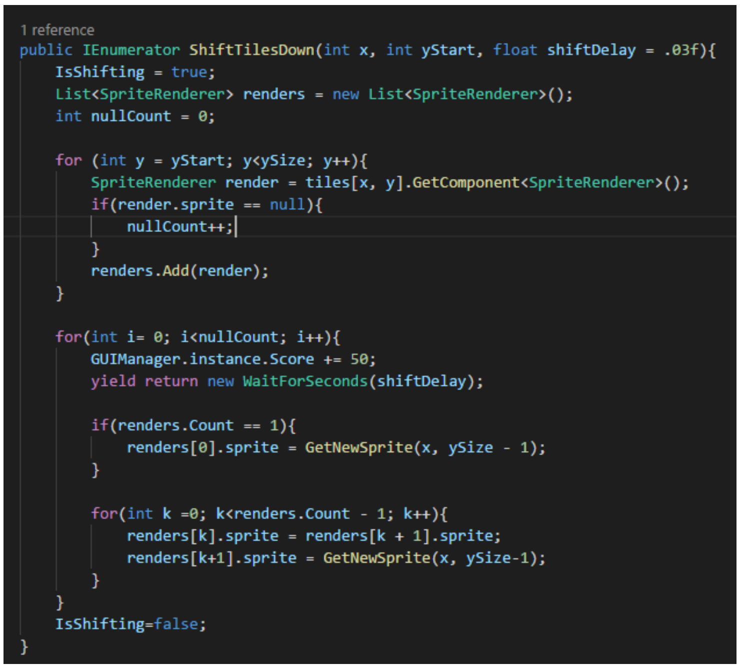
Figure 5. Changes to OnMouseDown method.



```
3 references
public bool ClearAllMatches(){
    if(render.sprite == null) return false;

    ClearMatch(new Vector2[2] {Vector2.left, Vector2.right});
    ClearMatch(new Vector2[2] {Vector2.up, Vector2.down});
    if(matchFound){
        render.sprite = null;
        matchFound = false;
        StopCoroutine(BoardManager.instance.FindNullTiles());
        StartCoroutine(BoardManager.instance.FindNullTiles());

        SFXManager.instance.PlaySFX(Clip.Clear);
        return true;
    }
    return false;


}
```

Figure 6. Changes to ClearAllMatches method.

The game also had a bug where the Tiles on the top row would sometimes not spawn. The fix provided by the user “hypnotistdk” fixes this issue in the ShiftTilesDown method and is applied in figure 7.



```
1 reference
public IEnumerator ShiftTilesDown(int x, int yStart, float shiftDelay = .03f){
    IsShifting = true;
    List<SpriteRenderer> renders = new List<SpriteRenderer>();
    int nullCount = 0;

    for (int y = yStart; y<ySize; y++){
        SpriteRenderer render = tiles[x, y].GetComponent<SpriteRenderer>();
        if(render.sprite == null){
            nullCount++;
        }
        renders.Add(render);
    }

    for(int i= 0; i<nullCount; i++){
        GUIManager.instance.Score += 50;
        yield return new WaitForSeconds(shiftDelay);

        if(renders.Count == 1){
            renders[0].sprite = GetNewSprite(x, ySize - 1);
        }

        for(int k =0; k<renders.Count - 1; k++){
            renders[k].sprite = renders[k + 1].sprite;
            renders[k+1].sprite = GetNewSprite(x, ySize-1);
        }
    }
    IsShifting=false;
}
```

Figure 7. Fixing the Tile spawn bug.

After these changes, the game fulfilled all the conditions set for the initial prototype and learning Clickteam Fusion could start.

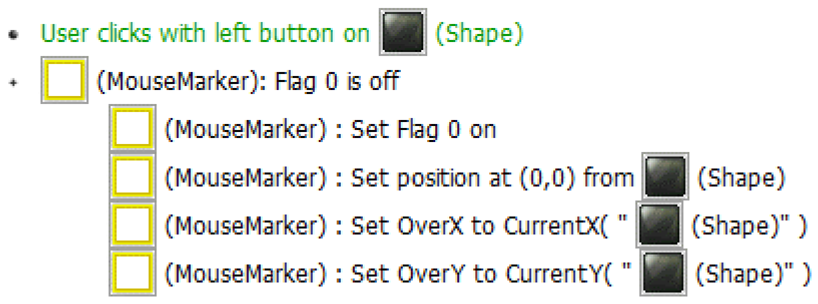
4.3 Clickteam Fusion Prototype

The development process for the Fusion prototype started completely from scratch. Within a few days, the default tutorials for the software were done to achieve a basic understanding of the application before starting the development. In these tutorials, the user gets to create clones of popular games such as Breakout and Flappy Bird. These tutorials gave an introduction on how to use many popular game development concepts inside Fusion, such as creating menus, implementing adjustable values, player controls, using the physics engine, handling collision et cetera.

The first impressions on Fusion development were split. On the other hand, doing things like basic menu functions or playing voice samples is very easy and effortless with Fusion. For comparison, when I have developed prototypes with Unity before, the sound system was not very easy to understand and took some time to figure out. In Fusion, there are not standard kind of methods or functions that are called each time a certain action is wanted to be activated, at least in the usual programming manner. Instead, the basic block of code in Fusion is called “event”, and an event always has a condition. The basic way of calling actions in Fusion is based on checking the states in the game constantly, and if a state matches to the condition of an event, the code inside the event block is ran instantly.

Regular functions can be emulated through “loops”. Unlike in basic object-oriented programming, if there are multiple objects of the same “class”, the wanted method cannot just be invoked through the reference of a specific object. If it is necessary to call an event on an object without directly interacting with it and thus singling it out for the scope of the event, a loop can be called that, for example, matches a value that is set to a variable of an object. There are some instances with Fusion where this feels quite odd or requires certain workarounds in order to make sure that the correct objects are being used in the event.

Here are some examples (Figure 8, 9) of Fusion events from the finished Fusion prototype. These examples will be easier understood when the whole context of the game is clear.



```
• User clicks with left button on (Shape)
+ (MouseMarker): Flag 0 is off
      (MouseMarker) : Set Flag 0 on
      (MouseMarker) : Set position at (0,0) from (Shape)
      (MouseMarker) : Set OverX to CurrentX( " (Shape)" )
      (MouseMarker) : Set OverY to CurrentY( " (Shape)" )
```

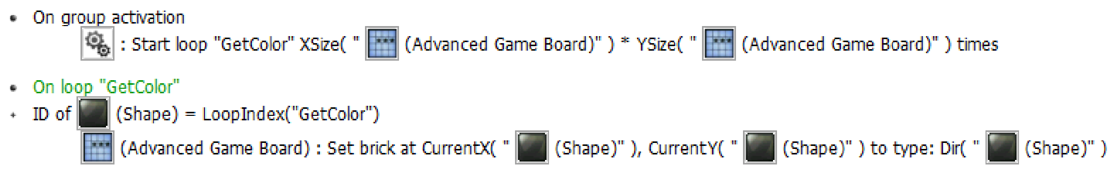
Figure 8. A Clickteam Fusion event.

This is what a basic block of code in Fusion looks like. The object that the event is called upon is determined by the first condition, in which the uses clicks a Shape object. Through which object is clicked, Fusion locks in the specific instance of the Shape object to which all the code in the event applies to.

In this example, there are two conditions for the event, and the Flag 0 variable for object MouseMarker needs to also be off. Flags in Fusion are variables that are present for each object that is created, and they can be either on or off, so they are commonly used for simple checks such as this. If the Flag was on, the event could not be called and very likely another block of code in the program would be activated instead. Because it is not specified in any way, which MouseMarker object the event is carried out for, it will automatically be done for all of them. In this example, there exists only one MouseMarker object, so this is acceptable.

In activation of the event, first the flag for the MouseMarker is changed from off to on. Then the position of the MouseMarker object is changed to the exact same coordinates as the chosen Shape object, with (0,0) signaling the offset from the object in X and Y coordinates. After this, the OverX and OverY variables of MouseMarker are changed to those of the CurrentX and CurrentY variables of the chosen shape. These variables keep track of where in the 8x8 sized grid each object currently resides, as this information will be required in the game logic.

Figure 9 is an example of a loop in Fusion. The events in Fusion can be separated into “groups” which can be disabled or enabled when required. If a group is disabled, none of the events inside it will be ever considered for triggering.



```
• On group activation
      : Start loop "GetColor" XSize( " (Advanced Game Board)" ) * YSize( " (Advanced Game Board)" ) times

• On loop "GetColor"
+ ID of (Shape) = LoopIndex("GetColor")
      (Advanced Game Board) : Set brick at CurrentX( " (Shape)" ), CurrentY( " (Shape)" ) to type: Dir( " (Shape)" )
```

Figure 9. Loops in Fusion.

Here it can be seen that a line of code is triggered when the group is changed to enabled from disabled. The first line of code will start the loop with the name “GetColor” a set amount of times, which is determined by the XSize and YSize variables of the object Advanced Game Board. The Advanced Game Board object is incredibly important for the game and its functions and the term “brick” will be inspected in greater detail later.

The second block of code is activated as the loop is initialized, and it will also take a specific Shape object through asking for a Shape object with the same ID as the current index of the GetColor loop. This will trigger the “set brick” functionality for the place in the game grid with the location corresponding to the CurrentX and CurrentY values of the Shape singled out by the condition.

The set brick function will then set the type of the brick to the same integer value that is determined by the current direction of the Shape object. If the Shape is, for example, pointing to the direction of 2 degrees, the value of the brick in the location of the Shape will become 2. In Fusion, storing different visual representations of an object in the direction angles is a very common practice, and does not necessarily mean that the actual direction of the object will be any different. So basically, this function goes through all the Shape objects and assigns the correct brick type to each brick on the grid.

4.3.1 Advanced Game Board

When going through tutorials and practicing Fusion, I discovered Fusion Shapes, a ClickTeam store product which included a full-fledged match three styled game and the source code. This game would be the source of some of the logic in the final game. [9]

After I started delving into how this game was created, a plugin to make developing this specific kind of genre was introduced. This plugin was the Advanced Game Board ([10]). The idea of this plugin is to provide an array to place the objects of the game in, while providing readymade functionality for moving the game objects and searching for connections created with the bricks, allowing for effective board manipulation. The plugin had to be downloaded through the Fusion plugin manager (Figure 13).

This sounded very intriguing and was a no-brainer to be chosen as the backbone of the prototype. The plugin contained multiple example games of classic games recreated with the Advanced Game Board, such as Four of a Kind, Tic Tac Toe and Tetris. It also included a bare-bones match-three game (Figure 10).

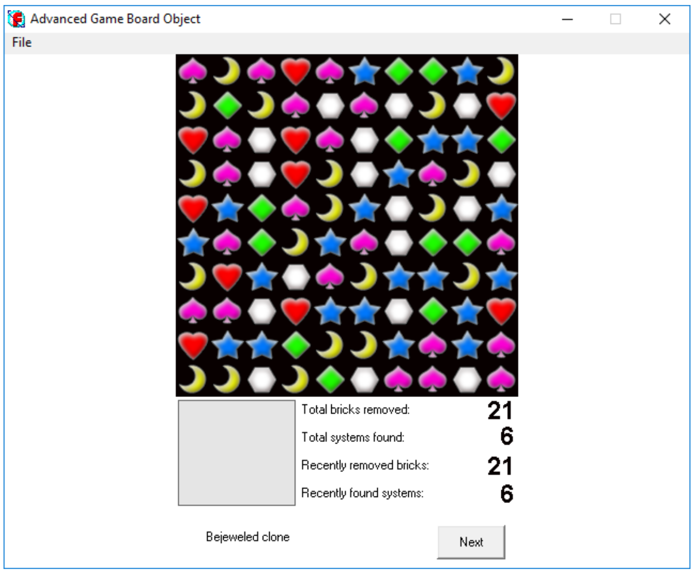

Advanced Game Board Object
File
Total bricks removed: 21
Total systems found: 6
Recently removed bricks: 21
Recently found systems: 6
Bejeweled clone
Next

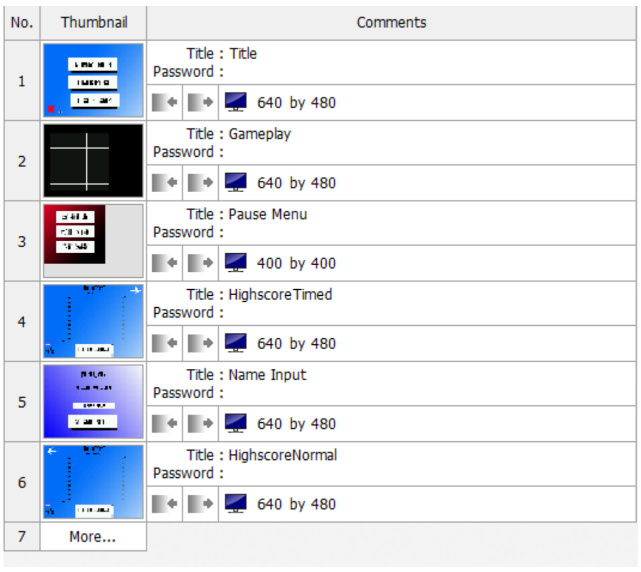
Figure 10. The AGB Match-three example.

The goal of the prototype was rather clear – there was no need to reinvent the wheel. I was going to create my own version of the AGB example from scratch, while also taking in inspiration from Fusion Shapes to improve functionality and add some other features. Things like an intricate scoring system and proper game over state recognition were to be left for the improved version of the game but making a prototype would start from this point.

4.3.2 The structure of game development in Clickteam Fusion

To fully understand what is done in the game development process and how it differs from the typical game engine, the basic structure of Fusion should be explained.

The editor of Fusion is divided into three parts: Storyboard Editor (Figure 11), Frame Editor (Figure 12) and the Event/Event List editor (Figures 15 and 16). The game is divided into frames, and each frame has its own events it will activate and follow. The developer can then move between these frames in their game through the events.



No.	Thumbnail	Comments
1		Title : Title Password : 640 by 480
2		Title : Gameplay Password : 640 by 480
3		Title : Pause Menu Password : 400 by 400
4		Title : HighscoreTimed Password : 640 by 480
5		Title : Name Input Password : 640 by 480
6		Title : HighscoreNormal Password : 640 by 480
7	More...	

Figure 11. The Storyboard Editor.

In this project, each frame is confined to the game window and does not extend outside of it, but in games which utilize scrolling such as platformers, the frames can be large, and only a small part of them is shown at once.

After creating a frame, it can be opened in Frame Editor to enter a drag and drop interface through which the developer can insert new objects and modify their settings through a properties window.

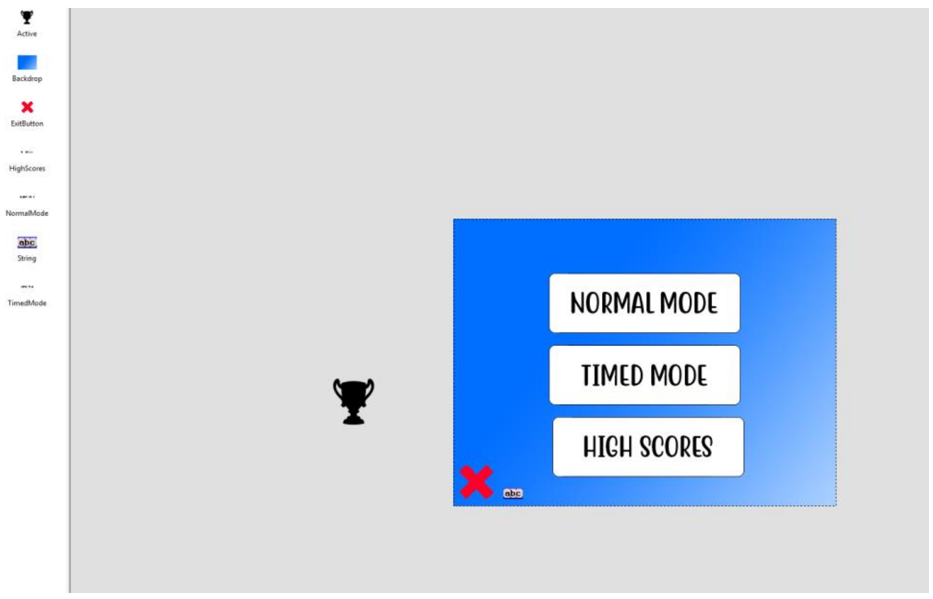


Figure 12. Frame Editor and its drag and drop interface.

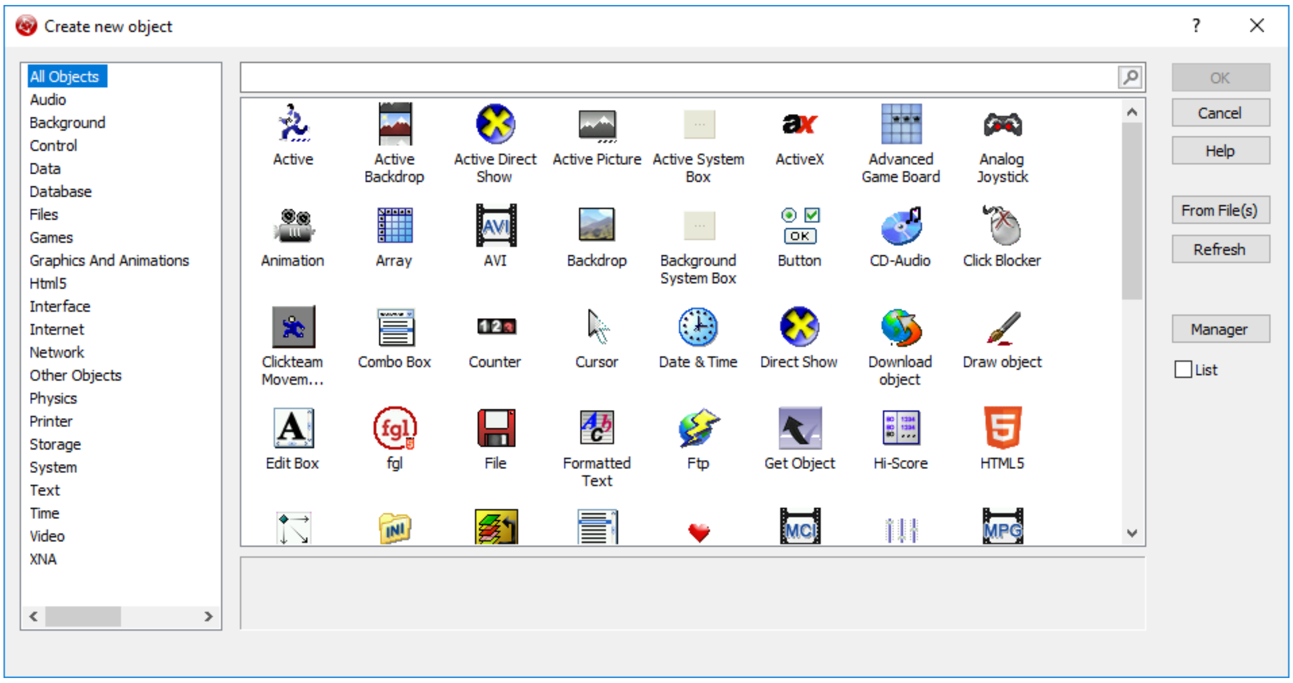
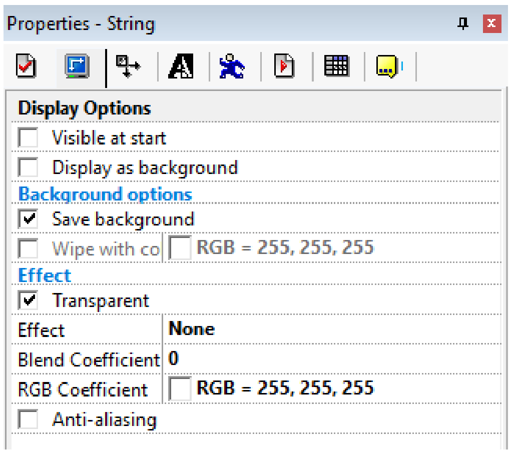


Figure 13. Creating new objects with several preinstalled object types. More can be found through the "Manager" button which opens the plugin manager, which was used to download Advanced Game Board.

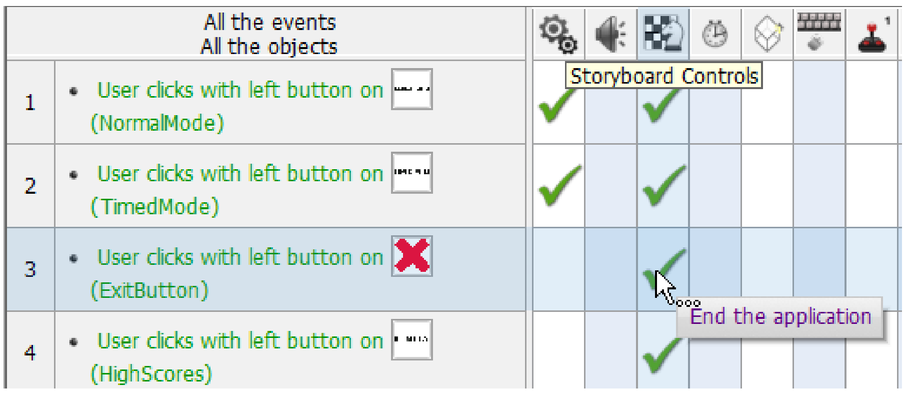
Each object has a properties window that can be used to modify their attributes and features (Figure 14)


Properties - String

Display Options	
☐ Visible at start	
☐ Display as background	
Background options	
☑ Save background	
☐ Wipe with co	☐ RGB = 255, 255, 255
Effect	
☑ Transparent	
Effect	**None**
Blend Coefficient	**0**
RGB Coefficient	☐ **RGB = 255, 255, 255**
☐ Anti-aliasing	

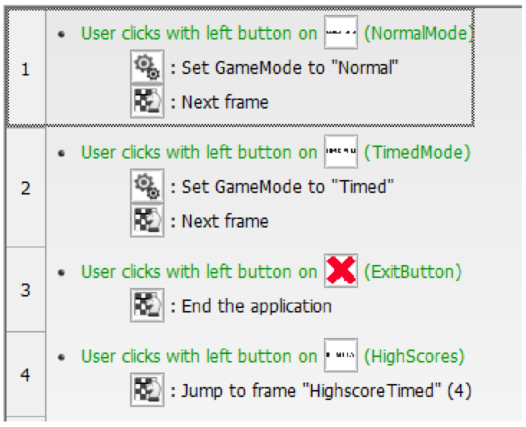
Figure 14. The properties window.

The Event Editor (Figure 15) is for programming the logic of the game, and it has two interfaces that can be used. The programming was mostly done with Event List Editor (Figure 16) as it is way clearer what is done in complex events through having the actions be all visible at once in text.



	All the events All the objects			Storyboard Controls				
1	• User clicks with left button on (NormalMode)	✓		✓				
2	• User clicks with left button on (TimedMode)	✓		✓				
3	• User clicks with left button on (ExitButton)			✓				
4	• User clicks with left button on (HighScores)			✓				

Figure 15. The standard Event Editor.



1	• User clicks with left button on (NormalMode) : Set GameMode to "Normal" : Next frame
2	• User clicks with left button on (TimedMode) : Set GameMode to "Timed" : Next frame
3	• User clicks with left button on (ExitButton) : End the application
4	• User clicks with left button on (HighScores) : Jump to frame "HighscoreTimed" (4)

Figure 16. The Event List Editor.

The initial prototype only consists of one frame, with the title “Gameplay”.

Figure 17 shows the finished prototype.

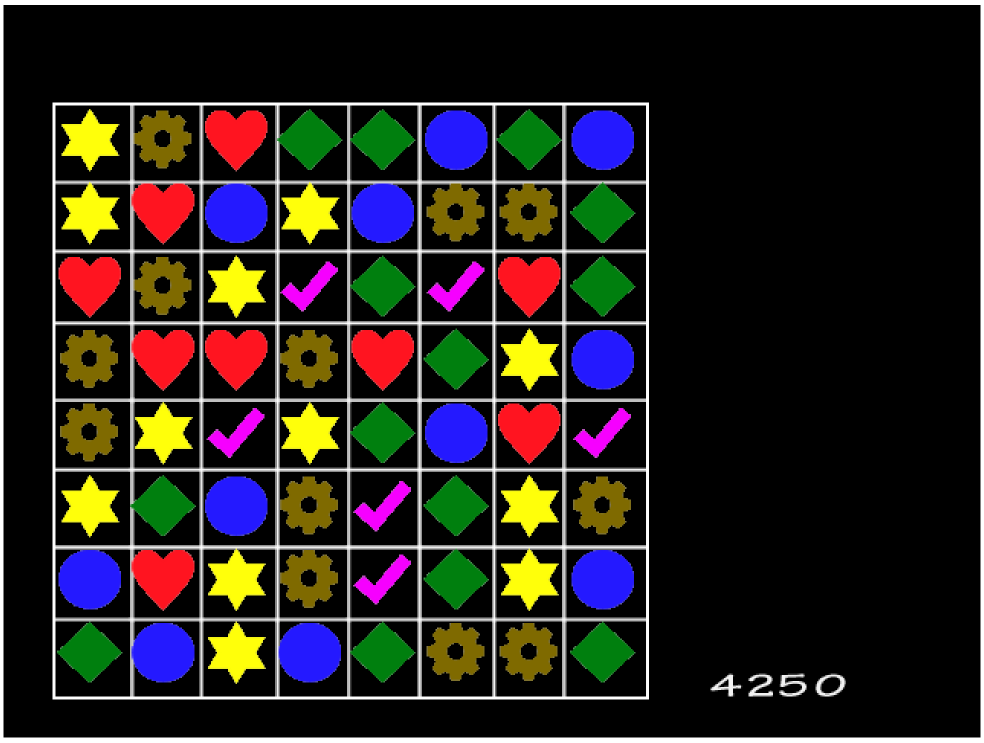


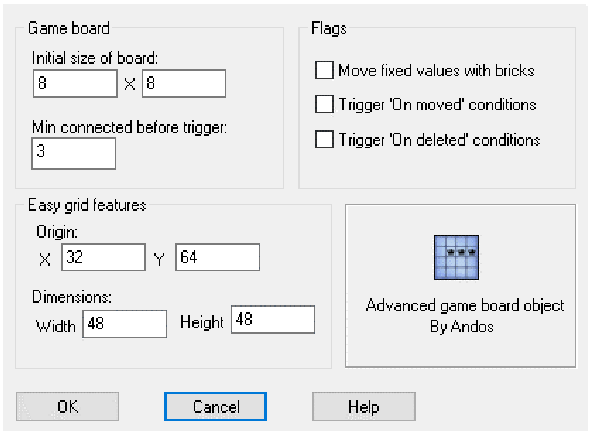
Figure 17. The finished prototype.

4.3.3 Objects and variables

Here the objects and variables used in the prototype are specified.

Advanced Game Board

The first thing that needs to be added to the game is the Advanced Game Board object that handles the object swapping and match finding (Figure 18 shows the settings window of the object). The initial size of the board is set to 8x8 as the game is meant to be played on a board with 64 objects layered out in a square. The origin of the board can be set to place the board wherever the developer wants, and this number should also not be ignored when designing the game and its events, as the information of the origin is important for certain matters. The dimensions boxes signify the size of each object in the grid.


Game board
Initial size of board: 8 X 8
Min connected before trigger: 3

Flags
☐ Move fixed values with bricks
☐ Trigger 'On moved' conditions
☐ Trigger 'On deleted' conditions

Easy grid features
Origin: X 32 Y 64
Dimensions: Width 48 Height 48

Advanced game board object
By Andos

Figure 18. The Advanced Game Board.

Shape

The primary playing object in the game is called Shape. There will always be 64 of these objects on the board, and the objects themselves will be stationary and exist in the same place for the whole game. What will change is the current "brick" value of each position in the Advanced Game Board, and the Shape object on top of each position will change its values to ensure that the image shown by the Shape will always correctly represent the value of the brick, and this will lead to the game working as intended.

The Shape object has six images assigned to six different direction values from 1-6, as there are six possible object types in the game. With five objects, the game would be very easy and with seven objects running out of possible moves would be very common. Very simple graphics for these objects were created. Figure 19 displays the image editor in Fusion.

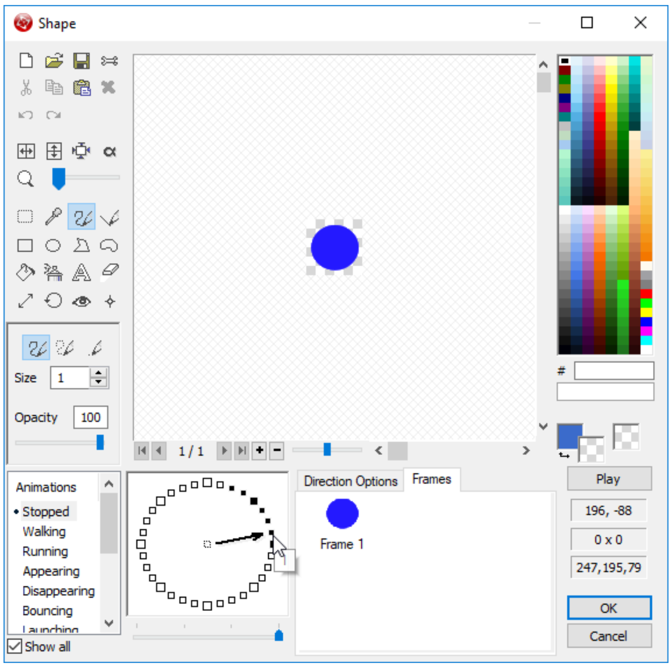


Figure 19. The Shape object opened in Fusion image editor.

Table 5. List of variables for Shape.

ID	The unique identifier of the Shape in the grid
Color	The current value of the Shape direction and image shown
CurrentX	The X location of the Shape in the Game Board
CurrentY	The Y location of the Shape in the Game Board

Array

A three-dimensional array object had to be placed in the game in order to be able to handle the logic of the initial playing field and then assigning its values to the Shape objects.

MouseMarker and SelectedMarker

These objects are simple yellow squares intended to give the player feedback on what piece is selected and where the second selection will be. The positions of these two objects are also used to determine if a move is legal, so the player cannot just swap any two squares at will. The act of locking in a piece and hovering another is shown in Figure 20.

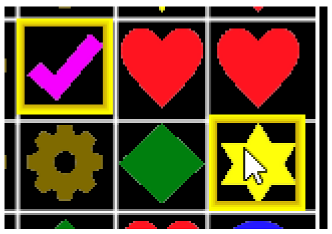


Figure 20. MouseMarker and SelectedMarker.

Table 6. Variables for MouseMarker.

OverX	Signifies in what X position of the grid is the Marker placed to.
OverY	Signifies in what Y position of the grid is the Marker placed to.

These values will be assigned to the functions of the Advanced Game Board when performing a swap to be able to tell what the first piece of the swap is.

Counter_Score

A default counter included in Fusion. This counter will hold the value of the current score. There will be no more advanced scoring system in place in the prototype, and this value will simply tick up by 50 every time a piece is destroyed on the board.

Timer

One last object present in the prototype is a simple timer that will be increased to a certain value whenever matches are created. This timer will then tick down fast, and some functions in the game will require this timer to be 0, such as adding new pieces to the board from the top.

The prototype also uses a single Active object (default object in Fusion) in order to use its flag property for some decisions. In hindsight this flag did not need to be on its own object, and it does not exist in the later versions.

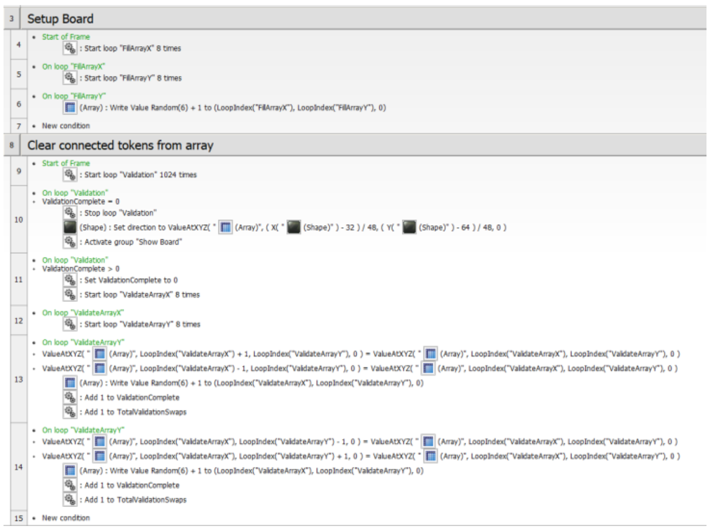
Table 7. Global variables in the prototype.

ShowGameboard	This value is ran from 0 to 65 in order to make all objects appear one at a time in the start of the game.
ValidationComplete	This is used when creating the initial game board. Before running the code that checks the board array for matches, this will be made 0. If a change must be made, this will tick up, so if the board is fine without any instant matches, this number will stay as 0. In the beginning of the loop it is checked if this number is 0, or more than 0. If it is 0, it means that during the previous iteration of the loop no problems were found, and the game can start.

There is a lot than can be done using global variables, but they were not really utilized in this prototype. A few variables were used for debugging purposes but none of them are important for the functionality of the game.

4.3.4 Generating the board

As the original example did not have a proper way of generating the board, I studied the code in Fusion Shapes for the logic. The code does exactly what is needed (Figure 21).



```
3  Setup Board
4  • Start of Frame
         : Start loop "FillArrayX" 8 times
5  • On loop "FillArrayX"
         : Start loop "FillArrayY" 8 times
6  • On loop "FillArrayY"
         (Array) : Write Value Random(6) + 1 to (LoopIndex("FillArrayX"), LoopIndex("FillArrayY"), 0)
7  • New condition
8  Clear connected tokens from array
9  • Start of Frame
         : Start loop "Validation" 1024 times
10 • On loop "Validation"
   • ValidationComplete = 0
         : Stop loop "Validation"
         (Shape) : Set direction to ValueAtXYZ( " (Array)", ( X( " (Shape)" ) - 32 ) / 48, ( Y( " (Shape)" ) - 64 ) / 48, 0 )
         : Activate group "Show Board"
11 • On loop "Validation"
   • ValidationComplete > 0
         : Set ValidationComplete to 0
         : Start loop "ValidateArrayX" 8 times
12 • On loop "ValidateArrayX"
         : Start loop "ValidateArrayY" 8 times
13 • On loop "ValidateArrayY"
   • ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX") + 1, LoopIndex("ValidateArrayY"), 0 ) = ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY"), 0 )
   • ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX") - 1, LoopIndex("ValidateArrayY"), 0 ) = ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY"), 0 )
         (Array) : Write Value Random(6) + 1 to (LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY"), 0)
         : Add 1 to ValidationComplete
         : Add 1 to TotalValidationSwaps
14 • On loop "ValidateArrayY"
   • ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY") - 1, 0 ) = ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY"), 0 )
   • ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY") + 1, 0 ) = ValueAtXYZ( " (Array)", LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY"), 0 )
         (Array) : Write Value Random(6) + 1 to (LoopIndex("ValidateArrayX"), LoopIndex("ValidateArrayY"), 0)
         : Add 1 to ValidationComplete
         : Add 1 to TotalValidationSwaps
15 • New condition
```

Figure 21. Setup Board and Clear connected tokens from array event groups.

Setup Board

This event group will run a loop in such a way that the Array object will be filled by 64 random values from 1 to 6. First all the Shapes with X = 0 (leftmost column) will be given a value, then the second column etc.

Clear connected tokens from array

The logic here is a bit complicated. The loop will be started 1024 times, which is arbitrary to ensure that the board will be complete, usually it takes under 5 iterations to arrive at a perfect board. ValidationComplete is set to 1 by default to ensure that the loop is ran for the first time, because the code checks if it is more than 0. Then the loops will be ran to give the validation functions the correct indexes to work with. Every time the code gets to the loop “ValidateArrayY”, the logic will look at values first at the both sides of the current value, and then the top and bottom values in the array and compare these values to the initial value.

For example, if the value of the array in the position 3, 2, 0 is the same as the values in positions 2, 2, 0 and 4, 2, 0, there are three values horizontally which contain the same value. This will lead to ValidationComplete ticking up and the current value decided by the loop indexes will be changed to another random value. After each run, the loops will be started again from zero, to ensure that the whole board is always checked. The TotalValidationSwaps global variable is here for debug purposes, so during testing it can be seen how many changes had to be made to the grid before arriving at the final array.

After the array passes through the loops with ValidationComplete staying at 0, the game is ready to start with the current board and the Validation loop is stopped. The directions of all Shape objects are set to corresponding values in the array. The offsets in the logic convert the pixelwise position of each Shape into an array. For example, the Shape that resides in the 1, 1 position of the grid is actually in the position 80, 112 of the frame, with the origin being the 0, 0 pixel of the Shape. So, if 32 is divided from 80 and 64 from 112 and both of these are divided by 48, resulting in the position 1, 1, 0 in the array, and its value will be assigned to the direction value of the Shape.

After this, the group "Show Board" (Figure 22) is activated, which will make the Shapes visible on the board.

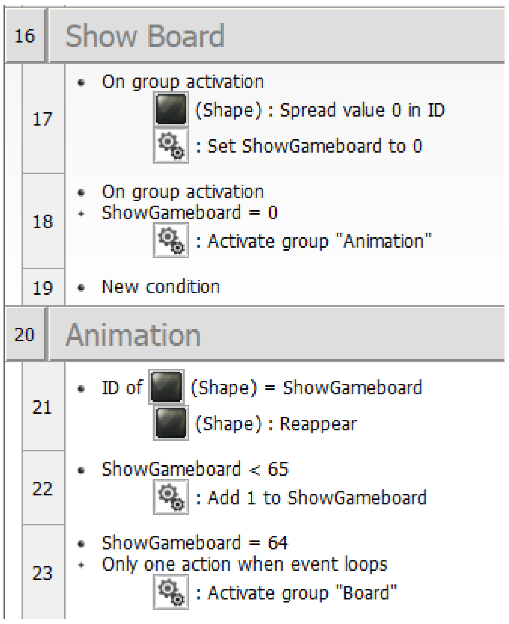
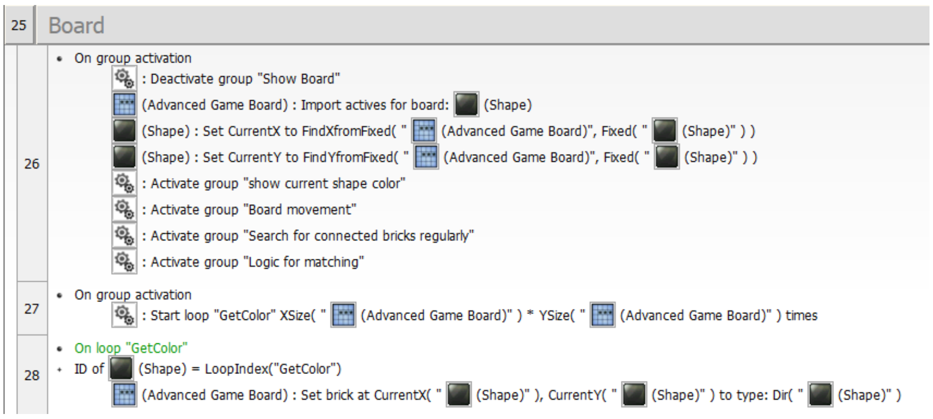

```
16  Show Board
17  • On group activation
          (Shape) : Spread value 0 in ID
          : Set ShowGameboard to 0
18  • On group activation
    + ShowGameboard = 0
          : Activate group "Animation"
19  • New condition
20  Animation
21  • ID of (Shape) = ShowGameboard
          (Shape) : Reappear
22  • ShowGameboard < 65
          : Add 1 to ShowGameboard
23  • ShowGameboard = 64
    + Only one action when event loops
          : Activate group "Board"
```

Figure 22. Show Board and Animation event groups.

This code will spread value 0 in the ID attribute of the Shape objects, giving them the IDs from 0 to 63 starting from the bottom right corner. The game board will be displayed by running the ShowGameboard global variable from 0 to 64. When the ID of a Shape matches the value of ShowGameboard, the Shape appears. All Shapes are set to be invisible in the start. After all Shapes are visible, the group “Board” (Figure 23) will be activated, and the game logic will be started.



```
Board
• On group activation
        : Deactivate group "Show Board"
        (Advanced Game Board) : Import actives for board: (Shape)
        (Shape) : Set CurrentX to FindXfromFixed( " (Advanced Game Board)", Fixed( " (Shape)" ) )
        (Shape) : Set CurrentY to FindYfromFixed( " (Advanced Game Board)", Fixed( " (Shape)" ) )
        : Activate group "show current shape color"
        : Activate group "Board movement"
        : Activate group "Search for connected bricks regularly"
        : Activate group "Logic for matching"
• On group activation
        : Start loop "GetColor" XSize( " (Advanced Game Board)" ) * YSize( " (Advanced Game Board)" ) times
• On loop "GetColor"
• ID of (Shape) = LoopIndex("GetColor")
        (Advanced Game Board) : Set brick at CurrentX( " (Shape)" ), CurrentY( " (Shape)" ) to type: Dir( " (Shape)" )
```

Figure 23. Board event.

Board

While activating the group Board, the group Show Board will simultaneously be turned off. First, the object Shape will be imported as the “active” for the Advanced Game Board, telling Advanced Game Board that the game is played through the Shape objects.

After this the CurrentX and CurrentY values of all Shapes are set through looking up their position according to the Advanced Game Board. The groups with game logic that can potentially disrupt with the game before it starts are turned on.

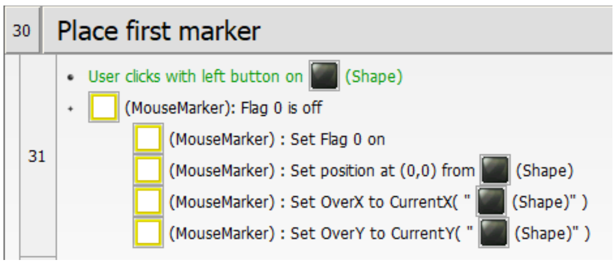
The loop “GetColor” is ran 64 times, from 0 to 63. For each brick, the brick type for Advanced Game Board is looked up from the direction value of each Shape. This is where CurrentX and CurrentY are required, without them the location that is wanted to be set could not be passed. The brick type must always be the same as the direction value of the Shape on top of it at each given point.

Now the game board has been initialized, and it is possible to start creating gameplay.

4.3.5 Selection of objects for swapping

First object

This step is very simple. Clicking on an object while the flag in MouseMarker is off triggers the event, with the MouseMarker flag signaling whether the MouseMarker currently exists on the board. The MouseMarker is initialized outside the game board, with this event moving it on top of the chosen Shape, with also adapting the OverX and OverY values from the CurrentX and CurrentY values of the Shape. The flag is also turned on to prevent this code from executing every time (Figure 24).

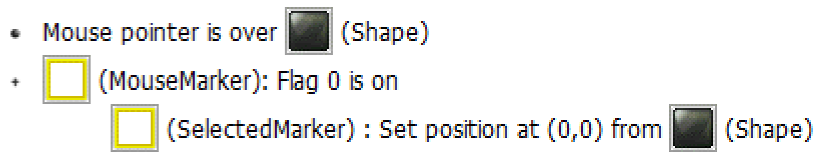


```
30  Place first marker
31  • User clicks with left button on (Shape)
    + (MouseMarker): Flag 0 is off
        (MouseMarker) : Set Flag 0 on
        (MouseMarker) : Set position at (0,0) from (Shape)
        (MouseMarker) : Set OverX to CurrentX( " (Shape)" )
        (MouseMarker) : Set OverY to CurrentY( " (Shape)" )
```

Figure 24. Placing first marker.

Selection of the second object

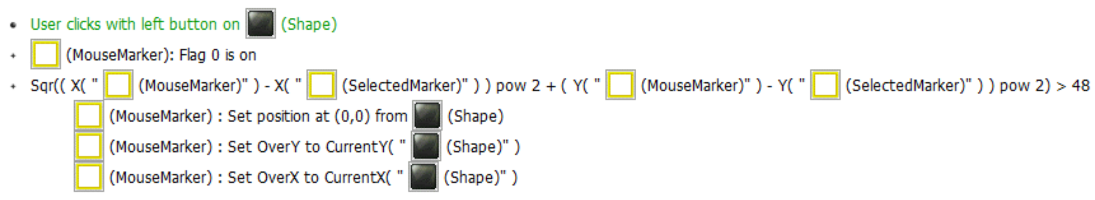
Whenever the flag in MouseMarker is on, the SelectedMarker object starts being locked to whichever Shape is currently hovered by the mouse cursor (Figure 25).



```
• Mouse pointer is over (Shape)
+ (MouseMarker): Flag 0 is on
    (SelectedMarker) : Set position at (0,0) from (Shape)
```

Figure 25. Displaying the marker at the location of the cursor.

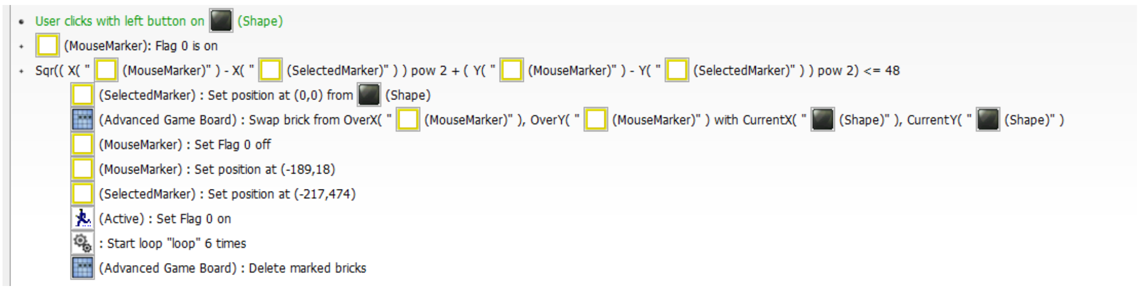
Whenever the user clicks on another Shape with the SelectedMarker enabled, the distance of the origin points of MouseMarker and SelectedMarker is calculated with the formula of distance between two points. If this is more than 48 pixels, the SelectedMarker is not on top of an adjacent Shape to MouseMarker. In this case, the selection will be transferred to the newly clicked Shape instead of initiating a swap (Figure 26).



```
• User clicks with left button on (Shape)
• (MouseMarker): Flag 0 is on
• Sqr(( X( " (MouseMarker)" ) - X( " (SelectedMarker)" ) ) pow 2 + ( Y( " (MouseMarker)" ) - Y( " (SelectedMarker)" ) ) pow 2) > 48
      (MouseMarker) : Set position at (0,0) from (Shape)
      (MouseMarker) : Set OverY to CurrentY( " (Shape)" )
      (MouseMarker) : Set OverX to CurrentX( " (Shape)" )
```

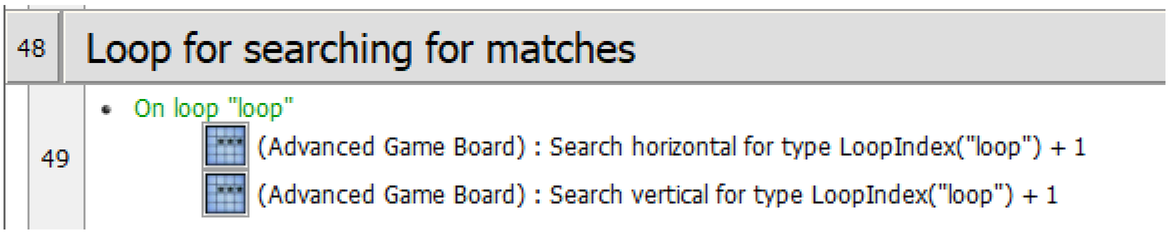
Figure 26. Changing the selection.

Figure 27 is the event that is initiated when the distance is less than 48 pixels, as in the SelectedMarker is on an adjacent Shape to the MouseMarker. The clicked Shape is chosen as the one the action is acted upon. The brick types of the chosen game board positions are swapped. The actual changing of the directions of the Shapes is done elsewhere in the code. After this MouseMarker flag is set off again and the marker objects are transferred outside the screen. In the prototype, an Active object was used to set a flag that is used with checking if the move results in matches. After this a loop is started that checks the board for each match type and triggers marking of matched bricks (Figure 28). These bricks are then deleted.



```
• User clicks with left button on (Shape)
• (MouseMarker): Flag 0 is on
• Sqr(( X( " (MouseMarker)" ) - X( " (SelectedMarker)" ) ) pow 2 + ( Y( " (MouseMarker)" ) - Y( " (SelectedMarker)" ) ) pow 2) <= 48
      (SelectedMarker) : Set position at (0,0) from (Shape)
      (Advanced Game Board) : Swap brick from OverX( " (MouseMarker)" ), OverY( " (MouseMarker)" ) with CurrentX( " (Shape)" ), CurrentY( " (Shape)" )
      (MouseMarker) : Set Flag 0 off
      (MouseMarker) : Set position at (-189,18)
      (SelectedMarker) : Set position at (-217,474)
      (Active) : Set Flag 0 on
      : Start loop "loop" 6 times
      (Advanced Game Board) : Delete marked bricks
```

Figure 27. Swapping adjacent bricks.



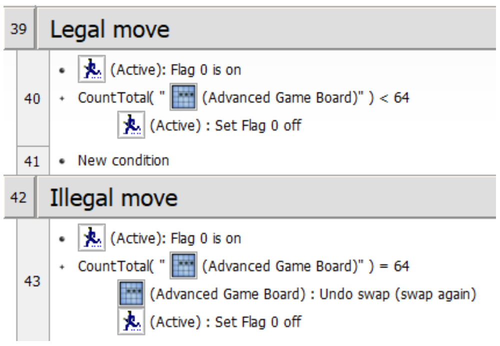
```
48  Loop for searching for matches
49  • On loop "loop"
          (Advanced Game Board) : Search horizontal for type LoopIndex("loop") + 1
          (Advanced Game Board) : Search vertical for type LoopIndex("loop") + 1
```

Figure 28. Loop for match searching.

Also note that if the player clicks on the first selected Shape again, this brick will swap with itself, resulting in cancelling the selection.

Checking for move legality

A swap should only be final if it results in matches in the game board. In the prototype, simple events are used to check this (Figure 29).



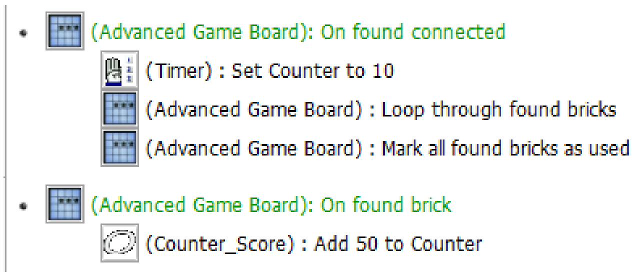
```
39  Legal move
40  • (Active): Flag 0 is on
    • CountTotal( " (Advanced Game Board)" ) < 64
          (Active) : Set Flag 0 off
41  • New condition
42  Illegal move
43  • (Active): Flag 0 is on
    • CountTotal( " (Advanced Game Board)" ) = 64
          (Advanced Game Board) : Undo swap (swap again)
          (Active) : Set Flag 0 off
```

Figure 29. Move legality test.

The Active flag is used to check whether the check is ongoing. After each swap, either of these events will trigger based on how many bricks exist on the game board. If this number is 64, it means that no match occurred, and the swap must be undone. This will be done in an instant in the prototype, resulting in that visibly nothing happens but the markers go away.

4.3.6 Deleting matched bricks

The previous loop is not enough to mark the bricks, it just triggers the condition "On found connected" which does this. It also sets the Timer to 10 to ensure that some other events do not run while connections are being made. The "On found brick" event is also triggered, in the prototype simply adding 50 points to the player's score (Figure 30).

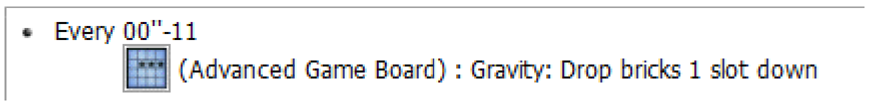


```
• (Advanced Game Board): On found connected
      (Timer) : Set Counter to 10
      (Advanced Game Board) : Loop through found bricks
      (Advanced Game Board) : Mark all found bricks as used
• (Advanced Game Board): On found brick
      (Counter_Score) : Add 50 to Counter
```

Figure 30. On found connected and on found brick conditions.

Dropping bricks down

The Gravity function of the Advanced Game Board object will deal with dropping the brick type downwards whenever there are deleted bricks (Figure 31). It will only do anything if there are bricks with the type 0 on the board, working as one would think. A delay condition is added to make the movement a bit slower, as it would trigger every frame otherwise.

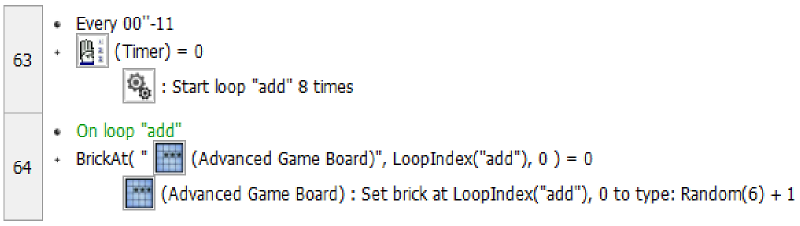


```
• Every 00"-11
        (Advanced Game Board) : Gravity: Drop bricks 1 slot down
```

Figure 31. Gravity in Advanced Game Board.

Adding new bricks to the top

The loop "add" is ran often to ensure that whenever everything possible has fallen, new bricks will be generated to the top which will also fall downwards until there are 64 bricks on the game board. This will simply go through the topmost line in the game board, making their brick type a random value from 1 to 6 if their brick type is 0, as in they do not exist (Figure 32).

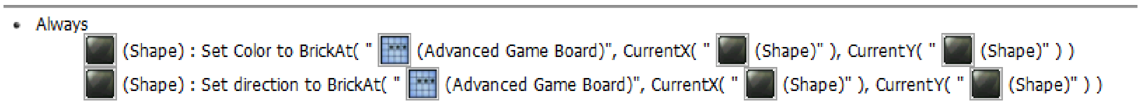


```
63  • Every 00"-11
    • (Timer) = 0
        : Start loop "add" 8 times
64  • On loop "add"
    • BrickAt( " (Advanced Game Board)", LoopIndex("add"), 0 ) = 0
        (Advanced Game Board) : Set brick at LoopIndex("add"), 0 to type: Random(6) + 1
```

Figure 32. Adding new bricks.

Updating the game board

All the previous events would be meaningless without this logic. All the Advanced Game Board does is move the brick type values around, it must be made sure that each Shape corresponds to the brick type currently in its position. This will simply activate for each Shape every frame to read the current brick type value in the Shape's position and set its direction to that value. The color is also set to the same value so in debugging it can be made sure that each Shape is always representing the correct brick type (Figure 33).



```
• Always
    (Shape) : Set Color to BrickAt( " (Advanced Game Board)", CurrentX( " (Shape)" ), CurrentY( " (Shape)" ) )
    (Shape) : Set direction to BrickAt( " (Advanced Game Board)", CurrentX( " (Shape)" ), CurrentY( " (Shape)" ) )
```

Figure 33. Updating the Shapes according to the game state.

This concludes the development of the Fusion prototype.

5 FURTHER DEVELOPMENT FROM A PROTOTYPE

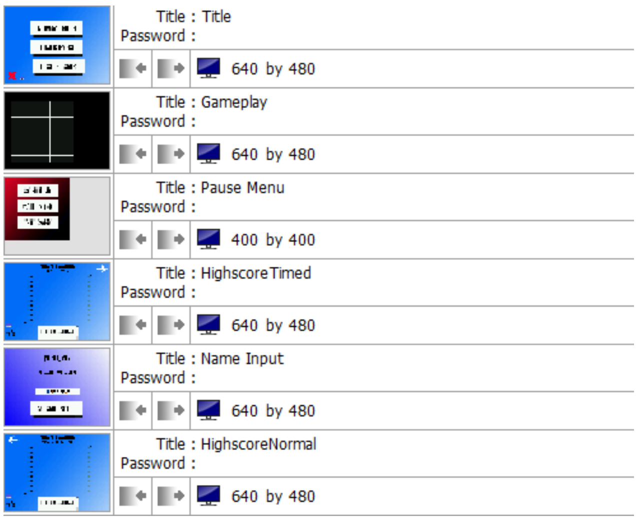
After the prototypes were finished, one of them was chosen for further development to make it resemble more of a finished game. The Fusion prototype was chosen because of the time that was invested into the engine and a will to find out more about its weaknesses and strengths.

Multiple things in the prototype were adjusted, added or remade, and the audiovisual design was created, along with the user interfaces and the necessary menus.

The list of things that were done is the following:

- Added main menu, pause menu and local leaderboards
- Two game modes, one of which is endless until the player runs out of moves and one is timed
- Improved game logic and a scoring system
- Graphics and sound design along with an ability to turn the sounds off

All the screenshots will be from the finished version, but during development the graphics were the last thing to be developed. Very simple placeholder graphics were used before that. Figure 34 displays the Storyboard of the finished game.



Title : Title
Password :
640 by 480

Title : Gameplay
Password :
640 by 480

Title : Pause Menu
Password :
400 by 400

Title : HighscoreTimed
Password :
640 by 480

Title : Name Input
Password :
640 by 480

Title : HighscoreNormal
Password :
640 by 480

Figure 34. The storyboard editor view of the game, showing the frames the game moves between.

5.1 List of new global variables

Table 8. List of new global variables.

GameMode	Determines the game mode. A global string.
ScoreMultiplier	Used in the scoring system. Bigger combo increases multiplier and gives more score
SwapInProgress	Used to check if a swap is currently happening
SwapTimer	Used to distinguish between matches created with swaps and with combos
UndoTimer	Used to add a delay to swapping bricks back after illegal move
MatchCount	Counts the combo amount
Matched	Holds the number of connected bricks after a successful match
MatchTimer	Used to get rid of the value in Matched right after it is used
ExitCurrentGame	Used to either exit to the main menu or exit the application from Pause Menu
GameTimer	Used to keep the current combo going and disallowing checking of swaps while it is above zero
SwapsLeft	Holds the current number of possible matches. Results in a game over when zero
ArrayCheckTempValue1	Used in checking the number of swaps left
ArrayCheckTempValue2	Used in checking the number of swaps left
NameEntered	Checks if a player name has been given

5.2 The main menu

The graphical style of the game in general was chosen to be minimalistic, based on blue and white gradients. The font for the buttons was chosen to be Quotable [11] which is available free for personal use. The main menu consists of three buttons which let you start a new game in either Normal or Timed mode or look at the local leaderboards. There is also a button in the corner that lets the player end the application. (Figure 35)

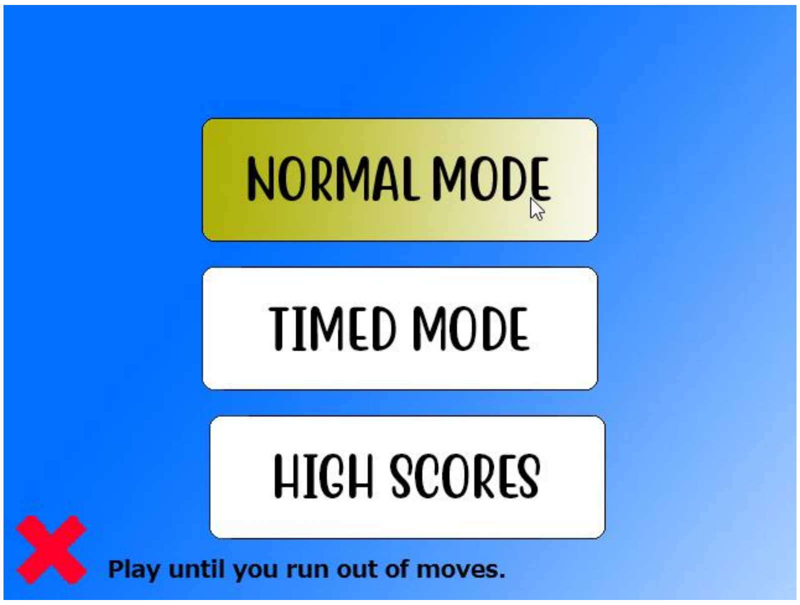


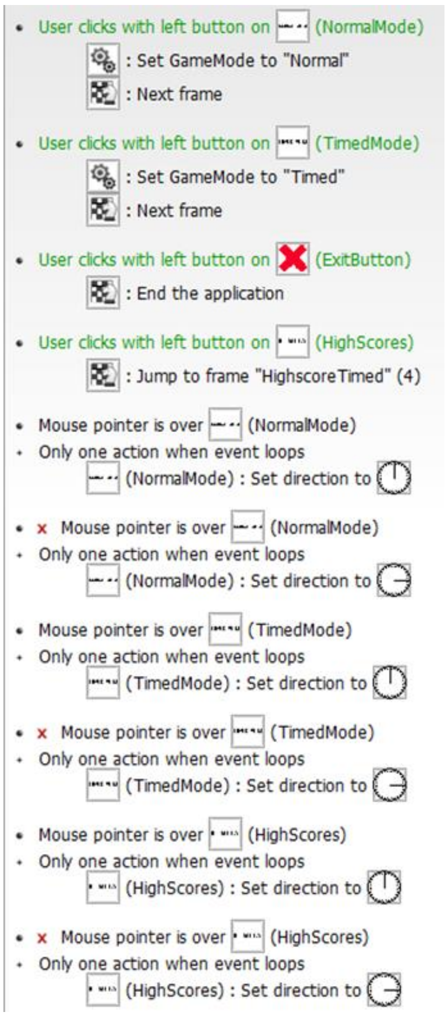
Figure 35. The main menu.

The buttons consist of two images, one of which is shown by default and a colored one when the button is hovered. There is also a text in the bottom which shows a description of the currently hovered option.

This text is produced through the String object in Fusion and does not use the Quotable font, as the font is not included in Windows and would not be displayed correctly.

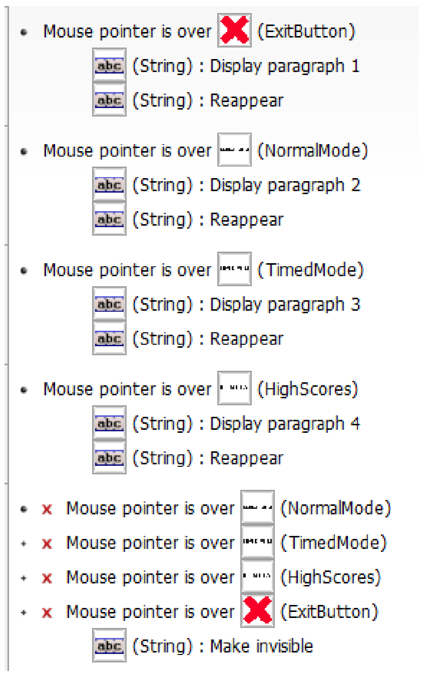
The code for the menu is very simple. Clicking on either of the new game buttons sets the GameMode global variable to the respective string, and advances the application to the next frame, being the Gameplay frame. Clicking the Exit button simply ends the application. The HighscoreTimed button moves the player to the leaderboard of the Timed game mode, from which the player can move to the leaderboard of the Normal game mode. Figure 36 displays the full code for the menu.

The two images associated with each button are set to different directions. This direction is determined from if the mouse cursor is hovering over the button or not. The negate feature of event conditions is used here, allowing the button to return to the white background when the mouse cursor leaves the button. So, if the button is hovered, the direction is set to 8, and if it is not, it is always set to 0. The string object in the bottom of the frame has 4 preset paragraphs that are displayed accordingly. If the cursor is not over any button in the interface, the string is not shown at all and is turned invisible. Hovering over any of the buttons changes the text of the object to describe the action of the button and simultaneously makes it visible (Figure 37).



- User clicks with left button on (NormalMode)
  - : Set GameMode to "Normal"
  - : Next frame
- User clicks with left button on (TimedMode)
  - : Set GameMode to "Timed"
  - : Next frame
- User clicks with left button on (ExitButton)
  - : End the application
- User clicks with left button on (HighScores)
  - : Jump to frame "HighscoreTimed" (4)
- Mouse pointer is over (NormalMode)
- Only one action when event loops
  - (NormalMode) : Set direction to
- x Mouse pointer is over (NormalMode)
- Only one action when event loops
  - (NormalMode) : Set direction to
- Mouse pointer is over (TimedMode)
- Only one action when event loops
  - (TimedMode) : Set direction to
- x Mouse pointer is over (TimedMode)
- Only one action when event loops
  - (TimedMode) : Set direction to
- Mouse pointer is over (HighScores)
- Only one action when event loops
  - (HighScores) : Set direction to
- x Mouse pointer is over (HighScores)
- Only one action when event loops
  - (HighScores) : Set direction to

Figure 36. The main menu logic.



```
• Mouse pointer is over (ExitButton)
        (String) : Display paragraph 1
        (String) : Reappear
• Mouse pointer is over (NormalMode)
        (String) : Display paragraph 2
        (String) : Reappear
• Mouse pointer is over (TimedMode)
        (String) : Display paragraph 3
        (String) : Reappear
• Mouse pointer is over (HighScores)
        (String) : Display paragraph 4
        (String) : Reappear
• x Mouse pointer is over (NormalMode)
• x Mouse pointer is over (TimedMode)
• x Mouse pointer is over (HighScores)
• x Mouse pointer is over (ExitButton)
        (String) : Make invisible
```

Figure 37. Changing the descriptive text.

5.3 Selecting a game mode

After clicking one of the buttons to start a new game, the game will be slightly different depending on the chosen mode. When initializing the game, no additional actions are required in Normal mode (Figure 38). If the game mode is Timed (Figure 39), counters displaying the time remaining are initialized and displayed in game (Figure 40). There is a counter that shows the time remaining in seconds and one that shows it as a bar that gradually goes down towards zero.

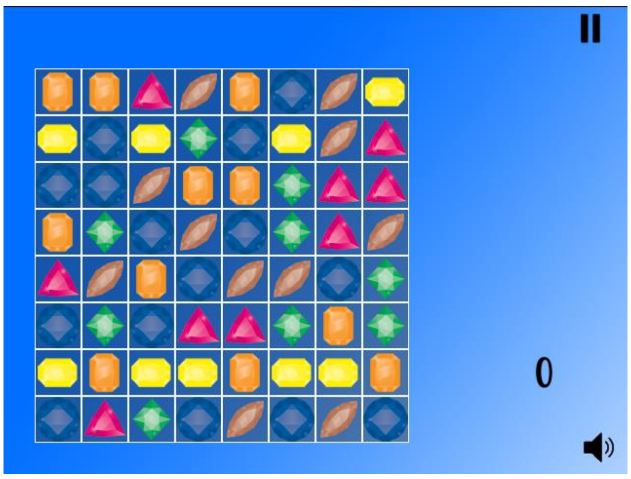


Figure 38. Normal Mode.

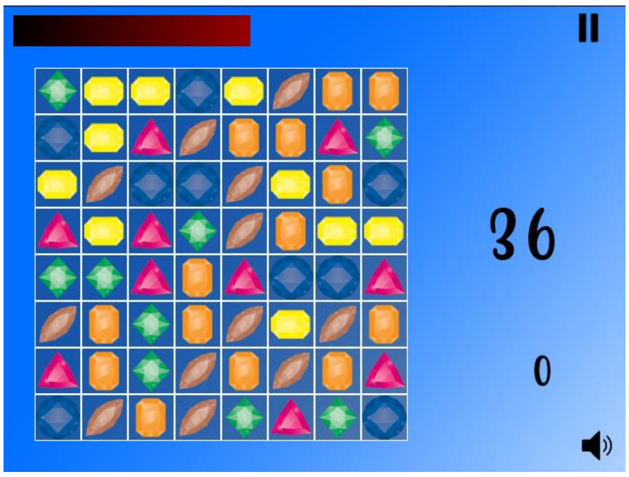
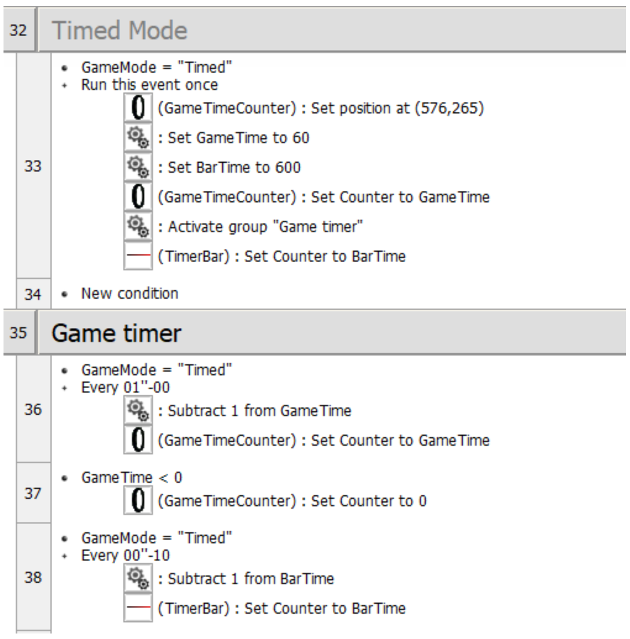


Figure 39. Timed Mode.



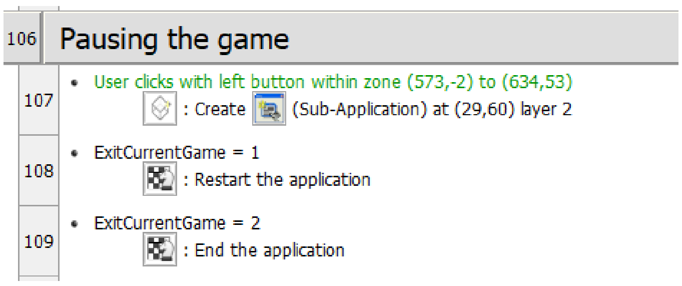
```
32  Timed Mode
33  • GameMode = "Timed"
    • Run this event once
          (GameTimeCounter) : Set position at (576,265)
          : Set GameTime to 60
          : Set BarTime to 600
          (GameTimeCounter) : Set Counter to GameTime
          : Activate group "Game timer"
          (TimerBar) : Set Counter to BarTime
34  • New condition
35  Game timer
36  • GameMode = "Timed"
    • Every 01"-00
          : Subtract 1 from GameTime
          (GameTimeCounter) : Set Counter to GameTime
37  • GameTime < 0
          (GameTimeCounter) : Set Counter to 0
38  • GameMode = "Timed"
    • Every 00"-10
          : Subtract 1 from BarTime
          (TimerBar) : Set Counter to BarTime
```

Figure 40. Code for Timed Mode.

Global variables are used for the timers because it was found during development that global variables are easy to track during debugging. The counter objects that have been added to the game board are just updated with the value of these variables whenever these values are updated.

5.4 Pause menu

By pressing the pause button in the upper right corner, the Pause Menu Sub-Application object can be accessed (Figure 41). This will halt the game actions and open the Pause Menu (Figure 42) on top of the Game Board. A Sub-Application object lets you run multiple frames on top of each other on Fusion.


106 Pausing the game

107
- User clicks with left button within zone (573,-2) to (634,53)
  : Create (Sub-Application) at (29,60) layer 2

108
- ExitCurrentGame = 1
  : Restart the application

109
- ExitCurrentGame = 2
  : End the application

Figure 41. Logic for pausing the game.

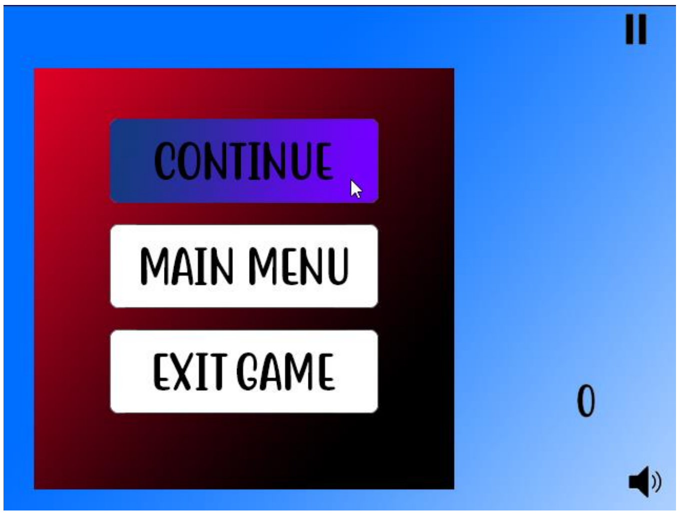
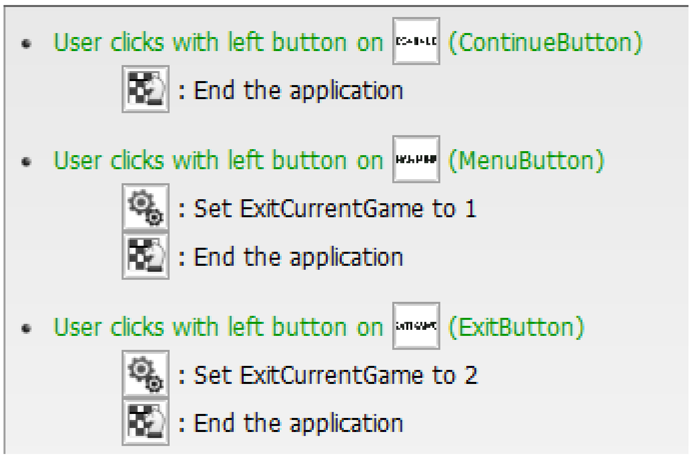


Figure 42. The game with the pause menu opened.

Clicking around the area of the Pause Button (this is better than assigning the event to the button itself because of its uneven shape) will open the Sub-Application that is set to the Pause Menu frame. It is placed on top of the gameplay to avoid foul play.

All the Pause Menu application does is modify the ExitCurrentGame global variable. If the user clicks on the Continue button, all that happens is that the Sub-Application terminates itself and the game will continue. If ExitCurrentGame is set to 1, the application will be restarted, ending the current run and launching the game back to Main Menu. This is smooth and handily resets all the variables to default values. If ExitCurrentGame is set to 2, the whole application will be terminated instantly. Figure 43 shows the Pause Menu logic.



- User clicks with left button on (ContinueButton)
  - : End the application
- User clicks with left button on (MenuButton)
  - : Set ExitCurrentGame to 1
  - : End the application
- User clicks with left button on (ExitButton)
  - : Set ExitCurrentGame to 2
  - : End the application

Figure 43. The Pause Menu logic.

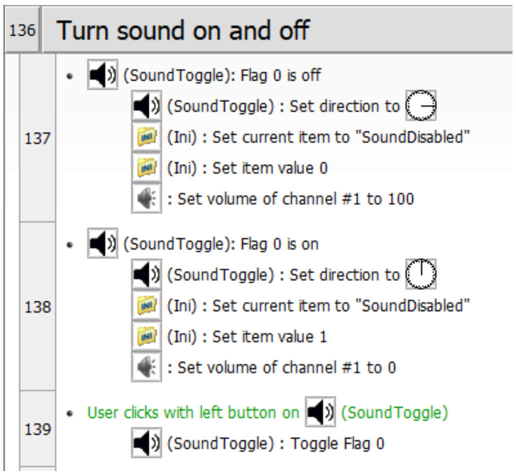
During the making of this thesis, a bug was found in the Sub-Application object: if the game window was moved to the right from the initial position, the Sub-Application would spawn in an incorrect position. This is very ugly, and the pause menu should be remade to have all the pause menu objects inside the Gameplay frame and halting all the necessary gameplay functions manually.

I was trying to find a solution to this but found nothing. I also downloaded other applications using the Sub-Application and got the same results on my computer, leading to the conclusion that there probably is a bug or an incompatibility in the Sub-Application component. If the game window is not moved after starting the game, the Sub-Application works as intended and I will not be editing because of the late time of finding the bug.

5.5 Toggling sound

Some sounds were added to the game, and there should be an option to turn these sounds off. More on the sound design will be discussed later in the document.

The game uses a file called Settings.ini that will be saved in the AppData folder of the user on Windows. This file could contain all kinds of settings, but the only one currently being saved is the ability to turn the sound off and on. The sound can be toggled through the button in the lower left corner during gameplay. (Figure 44)

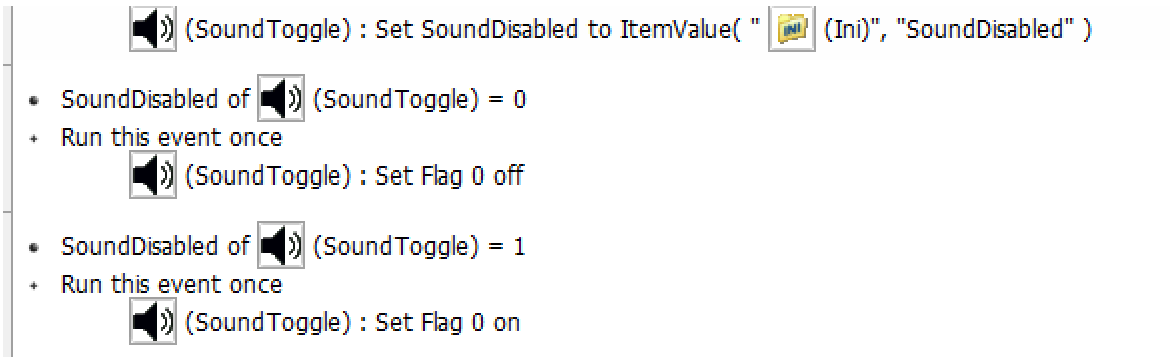


```
136 Turn sound on and off
137 • (SoundToggle): Flag 0 is off
        (SoundToggle) : Set direction to
        (Ini) : Set current item to "SoundDisabled"
        (Ini) : Set item value 0
        : Set volume of channel #1 to 100
138 • (SoundToggle): Flag 0 is on
        (SoundToggle) : Set direction to
        (Ini) : Set current item to "SoundDisabled"
        (Ini) : Set item value 1
        : Set volume of channel #1 to 0
139 • User clicks with left button on (SoundToggle)
        (SoundToggle) : Toggle Flag 0
```

Figure 44. Code for toggling sound.

The game checks every frame for the flag in the SoundToggle object and sets the image of the object properly and saves the setting in the setting file. If SoundDisabled is 1, the game will set the volume of channel 1 to 0. Whenever it is 0, the volume is 100. As all the sound samples will be played on channel 1 in this game, this will effectively mute all sounds while not interfering with music that could be added to the game (which would have its own way of getting turned off). The Toggle Flag event is very useful here, as it just changes the flag to 1 if it is 0 and vice versa.

Whenever the game is started, the setting file is read and the value of the flag of SoundToggle is initialized according to it (Figure 45). The flag cannot directly be set to a value so the value will be held by the SoundDisabled attribute created in the object itself temporarily.

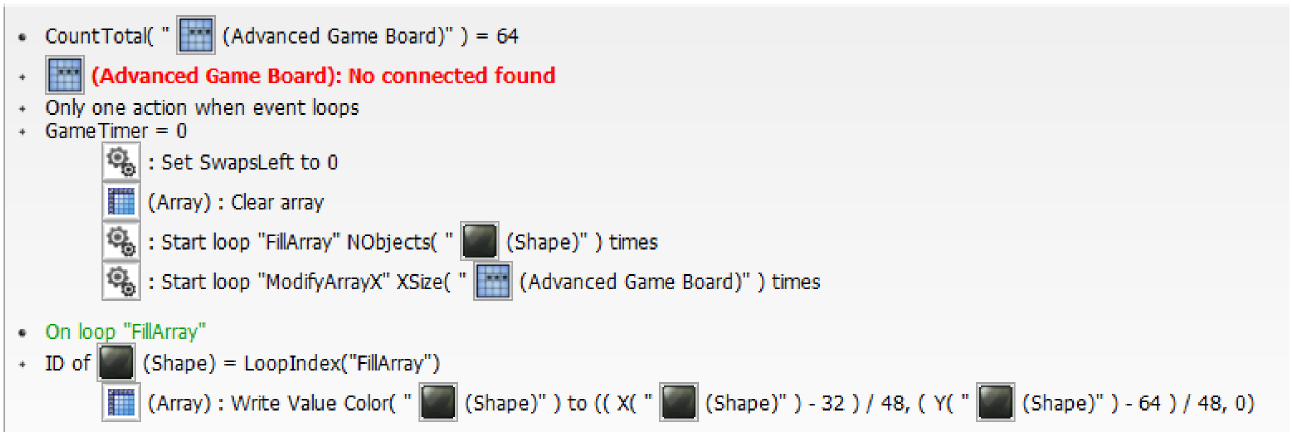


```
      (SoundToggle) : Set SoundDisabled to ItemValue( " (Ini)", "SoundDisabled" )

• SoundDisabled of (SoundToggle) = 0
+ Run this event once
      (SoundToggle) : Set Flag 0 off

• SoundDisabled of (SoundToggle) = 1
+ Run this event once
      (SoundToggle) : Set Flag 0 on
```

Figure 45. Reading the value of SoundToggle from a setting file.

5.6 Determining the number of swaps left on the board

This is the most complex functionality in the game. The game needs to keep track of the possible number of swaps on the board, so that it can end the game when there are none left. In this game, it is done through writing the state of the board into the Array object and simulating all the possible moves by swapping each object upwards and rightwards and checking if the swap results in a match. (Figure 46)



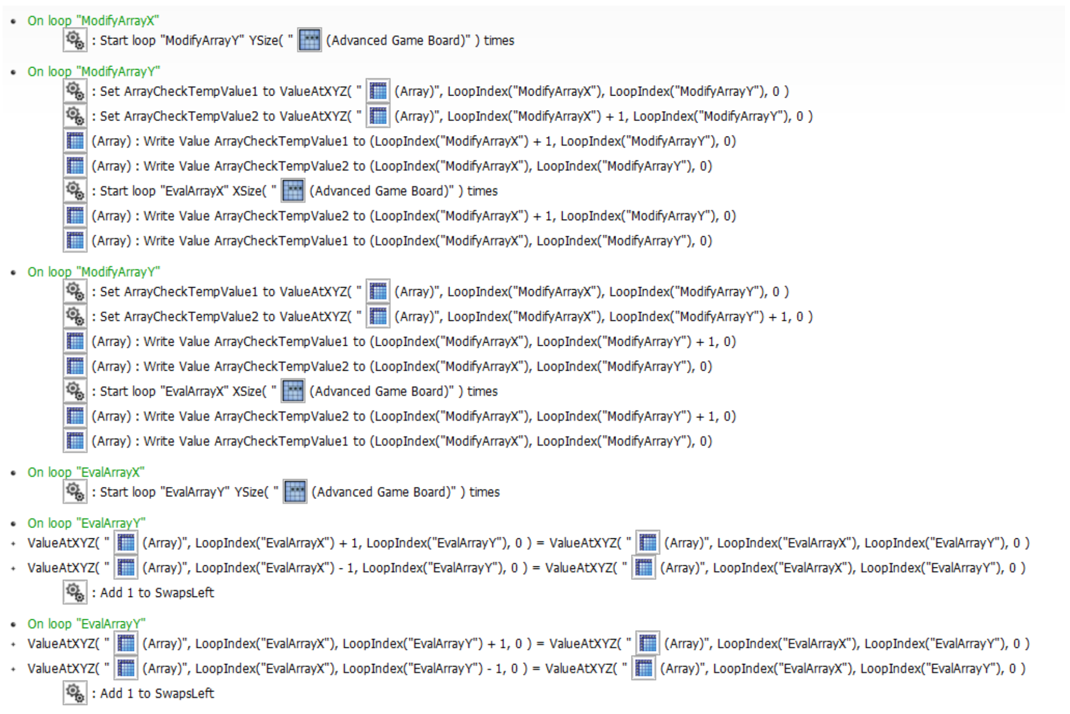
```
• CountTotal( " (Advanced Game Board)" ) = 64
+ (Advanced Game Board): No connected found
+ Only one action when event loops
+ GameTimer = 0
      : Set SwapsLeft to 0
      (Array) : Clear array
      : Start loop "FillArray" NObjects( " (Shape)" ) times
      : Start loop "ModifyArrayX" XSize( " (Advanced Game Board)" ) times

• On loop "FillArray"
+ ID of (Shape) = LoopIndex("FillArray")
      (Array) : Write Value Color( " (Shape)" ) to (( X( " (Shape)" ) - 32 ) / 48, ( Y( " (Shape)" ) - 64 ) / 48, 0)
```

Figure 46. Code of determining swaps left part 1.

The loops handling the functionality are ran every time GameTimer is 0 and there are 64 bricks on the board, meaning that no more matches are found currently, and everything is stationary. Counting the number of swaps would be very faulty if the board is not completely still. SwapsLeft is set to 0 before calculation and the Array object is cleared. FillArray is ran as many times as there are Shapes and the loop ModifyArrayX is ran eight times. Because the program is busy with these arrays, having SwapsLeft temporarily in 0 through this event will not cause other events to trigger with SwapsLeft = 0.

The board is written to the array through looping through the Shape objects and using their physical position to determine their place in the array and setting their Color value as their value in the array.

The logic is handled one column at a time, starting from 0, 0 to 0, 7. ModifyArrayY is ran once for each brick on the board, and contains two loops. These loops first swap the current brick and the one on the right to it in the first loop, and after this the current brick and the one above it in the second loop. The values of these two bricks are put into the ArrayCheckTempValue variables, and they are exchanged with each other in the array. After this, EvalArrayX and through it EvalArrayY are executed to check the whole board for swaps. EvalArrayY checks for the conditions first horizontally then vertically. If the value in the array in the location designated by the current loop indexes is identical of the value to the ones in X+1 and X-1 in the array, there is a horizontal match, and likewise if the value is identical to the ones in Y+1 and Y-1, there is a vertical match. In the case that either event is triggered, and a potential match is found, the SwapsLeft variable ticks. (Figure 47)



```
• On loop "ModifyArrayX"
        : Start loop "ModifyArrayY" YSize( " (Advanced Game Board)" ) times
• On loop "ModifyArrayY"
        : Set ArrayCheckTempValue1 to ValueAtXYZ( " (Array)", LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY"), 0 )
        : Set ArrayCheckTempValue2 to ValueAtXYZ( " (Array)", LoopIndex("ModifyArrayX") + 1, LoopIndex("ModifyArrayY"), 0 )
        (Array) : Write Value ArrayCheckTempValue1 to (LoopIndex("ModifyArrayX") + 1, LoopIndex("ModifyArrayY"), 0)
        (Array) : Write Value ArrayCheckTempValue2 to (LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY"), 0)
        : Start loop "EvalArrayX" XSize( " (Advanced Game Board)" ) times
        (Array) : Write Value ArrayCheckTempValue2 to (LoopIndex("ModifyArrayX") + 1, LoopIndex("ModifyArrayY"), 0)
        (Array) : Write Value ArrayCheckTempValue1 to (LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY"), 0)
• On loop "ModifyArrayY"
        : Set ArrayCheckTempValue1 to ValueAtXYZ( " (Array)", LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY"), 0 )
        : Set ArrayCheckTempValue2 to ValueAtXYZ( " (Array)", LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY") + 1, 0 )
        (Array) : Write Value ArrayCheckTempValue1 to (LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY") + 1, 0)
        (Array) : Write Value ArrayCheckTempValue2 to (LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY"), 0)
        : Start loop "EvalArrayX" XSize( " (Advanced Game Board)" ) times
        (Array) : Write Value ArrayCheckTempValue2 to (LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY") + 1, 0)
        (Array) : Write Value ArrayCheckTempValue1 to (LoopIndex("ModifyArrayX"), LoopIndex("ModifyArrayY"), 0)
• On loop "EvalArrayX"
        : Start loop "EvalArrayY" YSize( " (Advanced Game Board)" ) times
• On loop "EvalArrayY"
+ ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX") + 1, LoopIndex("EvalArrayY"), 0 ) = ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX"), LoopIndex("EvalArrayY"), 0 )
+ ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX") - 1, LoopIndex("EvalArrayY"), 0 ) = ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX"), LoopIndex("EvalArrayY"), 0 )
        : Add 1 to SwapsLeft
• On loop "EvalArrayY"
+ ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX"), LoopIndex("EvalArrayY") + 1, 0 ) = ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX"), LoopIndex("EvalArrayY"), 0 )
+ ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX"), LoopIndex("EvalArrayY") - 1, 0 ) = ValueAtXYZ( " (Array)", LoopIndex("EvalArrayX"), LoopIndex("EvalArrayY"), 0 )
        : Add 1 to SwapsLeft
```

Figure 47. Code of determining swaps left part 2.

After this the array is returned to its original state through assigning the values saved in the temporary values back to their original positions, and the second loop will work in a similar fashion to the other direction.

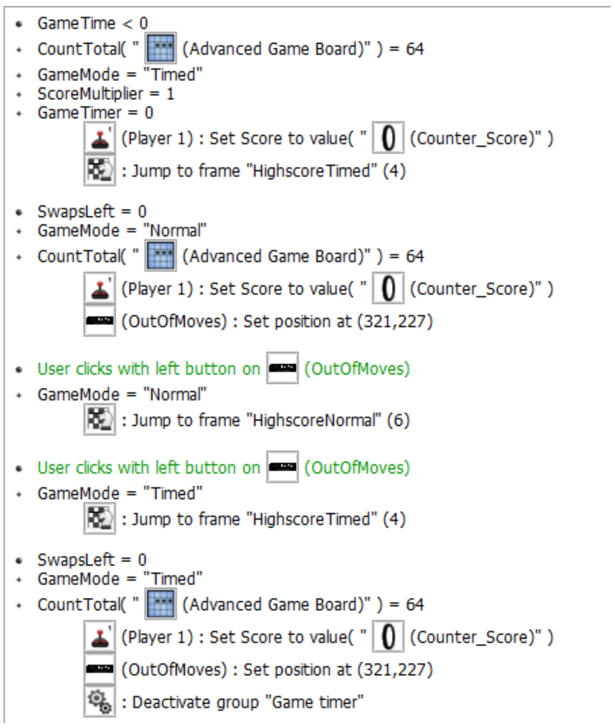
This function works properly and will always result in the SwapsLeft variable containing the number of possible swaps that can be made in the game.

5.7 Game over

The game needs to have an end condition so the score can be saved to the high score list. If GameMode is set to Normal, only running out of moves (SwapsLeft = 0) will trigger the end of the game. If GameMode is timed, running out of time is the primary reason for a game over, but running out of moves is also possible.

In Timed mode, the first condition to triggering a game over will look for the GameTime value to be under 0. This will let the game go on even if the timer is 0, adding more excitement to time running out instead of ending the game instantly. To help this, also if there is anything happening on the board while the time runs out, the conditions will let the board stabilize before ending the game. Triggering this event will set the Player 1 Score value to the value designated in the Counter_Score object, letting it be used to be saved on the high score list. In timed mode, there is no pop up to notify of the time running out, but the game jumps to the HighscoreTimed frame. (Figure 48)

In Normal mode, it is simply checked that SwapsLeft is 0 and the count of bricks is 64. This means there are no more possible moves on the board and the “OutOfMoves” object will be displayed to notify the player. Clicking this will transfer the player to the HighscoreNormal frame. Similar events are also given to the Timed mode. Figure 49 shows the player running out of moves in Normal mode.



```
• GameTime < 0
+ CountTotal( " (Advanced Game Board)" ) = 64
+ GameMode = "Timed"
+ ScoreMultiplier = 1
+ GameTimer = 0
        (Player 1) : Set Score to value( " (Counter_Score)" )
        : Jump to frame "HighscoreTimed" (4)

• SwapsLeft = 0
+ GameMode = "Normal"
+ CountTotal( " (Advanced Game Board)" ) = 64
        (Player 1) : Set Score to value( " (Counter_Score)" )
        (OutOfMoves) : Set position at (321,227)

• User clicks with left button on (OutOfMoves)
+ GameMode = "Normal"
        : Jump to frame "HighscoreNormal" (6)

• User clicks with left button on (OutOfMoves)
+ GameMode = "Timed"
        : Jump to frame "HighscoreTimed" (4)

• SwapsLeft = 0
+ GameMode = "Timed"
+ CountTotal( " (Advanced Game Board)" ) = 64
        (Player 1) : Set Score to value( " (Counter_Score)" )
        (OutOfMoves) : Set position at (321,227)
        : Deactivate group "Game timer"
```

Figure 48. Game over.

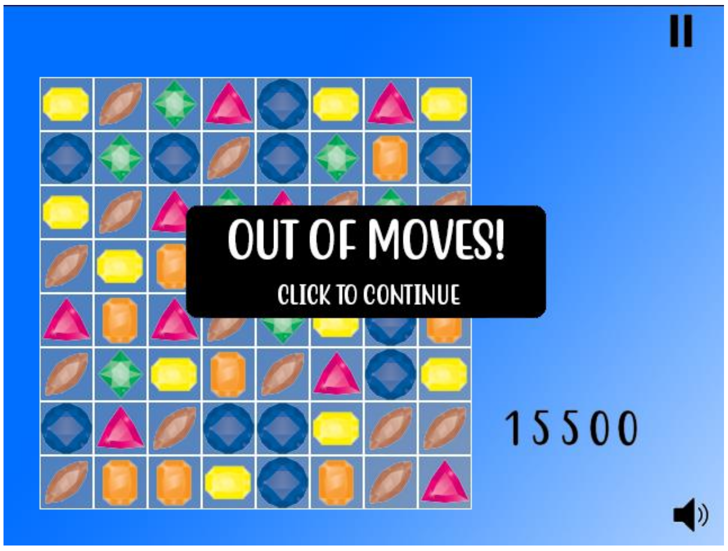
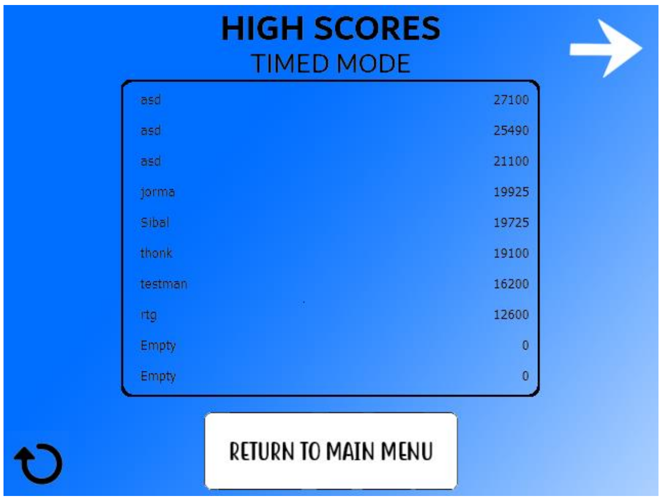


Figure 49. Game over in Normal mode.

5.8 The High score frames

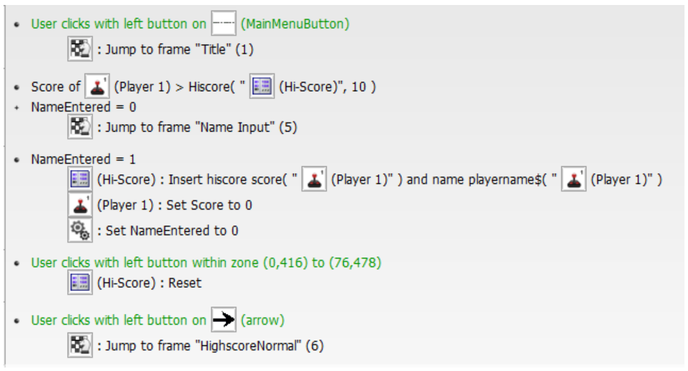
The game includes a high score frame for each game mode, HighscoreTimed and HighscoreNormal, as well as a frame called Name Input which is used to input the player name. The frames are identical with their logic, containing the leaderboard for their respective game mode as well as buttons to move to the other leaderboard, a button to reset the respective leaderboard and a button to return to the main menu. (Figure 50).



asd	27100
asd	25490
asd	21100
jorma	19925
Sibal	19725
thonk	19100
testman	16200
rtg	12600
Empty	0
Empty	0

Figure 50. The High score frame.

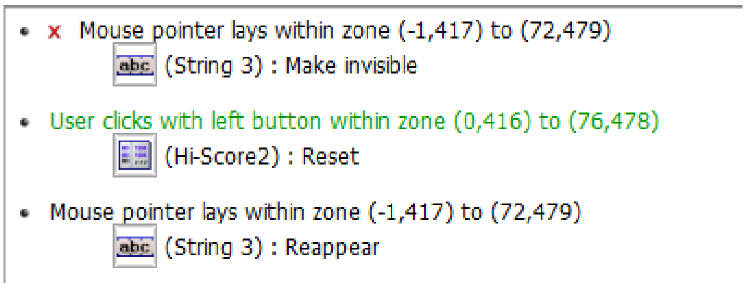
Figure 51 shows the code for the high score lists. If the score of the player is better than the one in the 10th place of the board, the player is prompted with a name input screen. This name will be saved along with the score the player achieved. The name input screen (Figure 53) will set the NameEntered variable to 1 and the score will be saved (Figure 54).



```
• User clicks with left button on (MainMenuButton)
      : Jump to frame "Title" (1)

• Score of (Player 1) > Hiscore( " (Hi-Score)", 10 )
• NameEntered = 0
      : Jump to frame "Name Input" (5)

• NameEntered = 1
      (Hi-Score) : Insert hiscore score( " (Player 1)" ) and name playername$( " (Player 1)" )
      (Player 1) : Set Score to 0
      : Set NameEntered to 0

• User clicks with left button within zone (0,416) to (76,478)
      (Hi-Score) : Reset

• User clicks with left button on (arrow)
      : Jump to frame "HighscoreNormal" (6)
```

Figure 51. High score code.

The button in the bottom left corner can reset the Hi-Score object used to save the leaderboard, and an informatory text will appear if the button is hovered. Because of the uneven shape of the image, zone recognition is used instead of the object itself. (Figure 52)



- x Mouse pointer lays within zone (-1,417) to (72,479)
  (String 3) : Make invisible
- User clicks with left button within zone (0,416) to (76,478)
  (Hi-Score2) : Reset
- Mouse pointer lays within zone (-1,417) to (72,479)
  (String 3) : Reappear

Figure 52. Resetting the Hi-Score object.

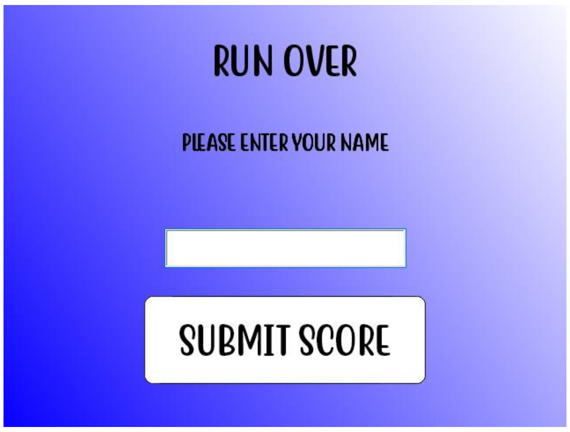


Figure 53. Name entry.

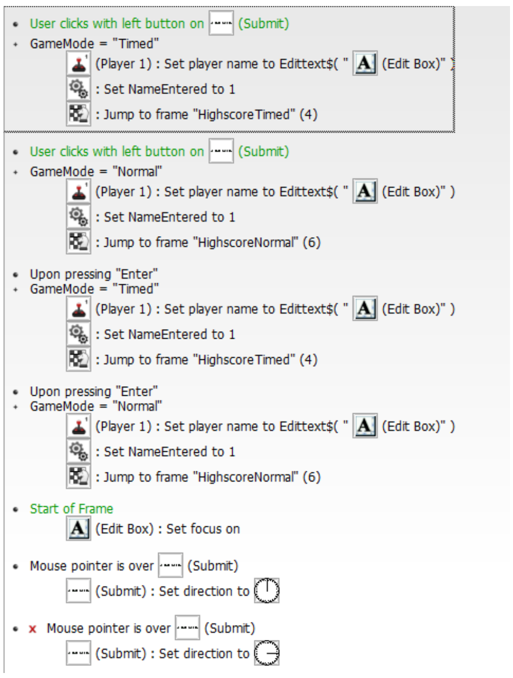

```
• User clicks with left button on (Submit)
+ GameMode = "Timed"
      (Player 1) : Set player name to Edittext$( " (Edit Box)" )
      : Set NameEntered to 1
      : Jump to frame "HighscoreTimed" (4)

• User clicks with left button on (Submit)
+ GameMode = "Normal"
      (Player 1) : Set player name to Edittext$( " (Edit Box)" )
      : Set NameEntered to 1
      : Jump to frame "HighscoreNormal" (6)

• Upon pressing "Enter"
+ GameMode = "Timed"
      (Player 1) : Set player name to Edittext$( " (Edit Box)" )
      : Set NameEntered to 1
      : Jump to frame "HighscoreTimed" (4)

• Upon pressing "Enter"
+ GameMode = "Normal"
      (Player 1) : Set player name to Edittext$( " (Edit Box)" )
      : Set NameEntered to 1
      : Jump to frame "HighscoreNormal" (6)

• Start of Frame
      (Edit Box) : Set focus on

• Mouse pointer is over (Submit)
      (Submit) : Set direction to

• x Mouse pointer is over (Submit)
      (Submit) : Set direction to
```

Figure 54. Code for Name Input frame.

The name entry screen has a textbox and button. The focus will be set when the frame is initialized, so that the player does not have to click on the box. After clicking the submit button or pressing Enter, player name is set, NameEntered is 1 and the frame transfers back to the correct leaderboard.

The name entry was also initially performed by a Sub-Application object but was changed after realizing its bugginess.

5.9 Design of the scoring system

The scoring system works using raw score values set for obtaining certain matches and a score multiplier that multiplies the score gained from each individual brick. It takes advantage of the following global variables: MatchTimer, Matched, MatchCount and GameTimer. Also, SwapTimer is used when determining the initial swap.

Whenever a swap is initiated, the SwapTimer is briefly set to 1. This is used to distinguish the initial swap. During the initial swap, the MatchCount cannot go above 1, as the combo system requires matches to be made with falling pieces. After this, SwapTimer ticks to 0, letting all future matches increase the MatchCount (Figure 55). This rewards the player for making big combos through falling bricks.

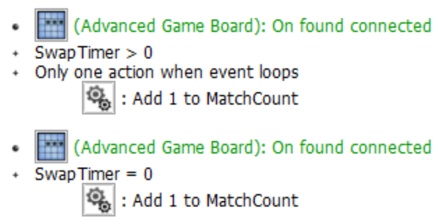
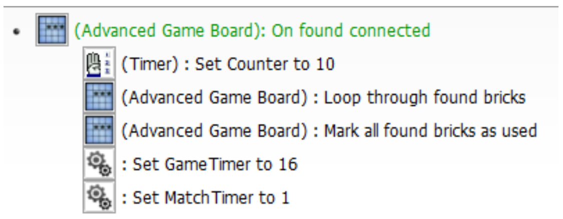

```
• (Advanced Game Board): On found connected
+ SwapTimer > 0
+ Only one action when event loops
        : Add 1 to MatchCount

• (Advanced Game Board): On found connected
+ SwapTimer = 0
        : Add 1 to MatchCount
```

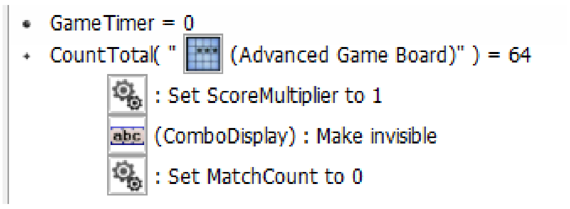
Figure 55. Distinguishing the initial swap.

Whenever a match is made (Figure 56), the GameTimer is set to 16 and MatchTimer to 1. The GameTimer is refreshed every time a match is made, and when it reaches zero, it means that the board has returned to the normal state and ScoreMultiplier is returned to 1 and MatchCount to 0 (Figure 57).



```
• (Advanced Game Board): On found connected
        (Timer) : Set Counter to 10
        (Advanced Game Board) : Loop through found bricks
        (Advanced Game Board) : Mark all found bricks as used
        : Set GameTimer to 16
        : Set MatchTimer to 1
```

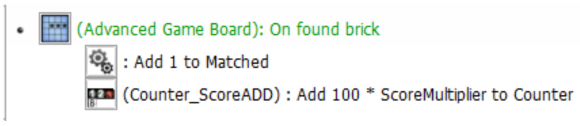
Figure 56. The event that is triggered every match.



```
• GameTimer = 0
• CountTotal( " (Advanced Game Board)" ) = 64
        : Set ScoreMultiplier to 1
        (ComboDisplay) : Make invisible
        : Set MatchCount to 0
```

Figure 5 Resetting scoring variables

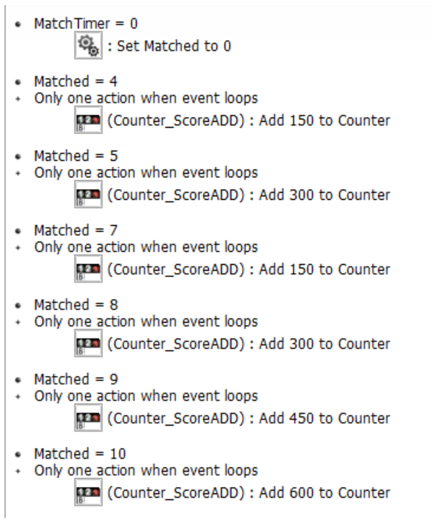
Whenever a brick is found in a connection, the value of Matched is increased by 1 and 100 is added to the score of the player, multiplied by the ScoreMultiplier (Figure 58). The value of Matched is used to determine four-of-a -kinds and five-of-a-kinds, giving a raw bonus point value to the score. After this, MatchTimer instantly ticks to 0, resetting the value of Matched. This way, Matched counts all the bricks found on the same frame.



```
• (Advanced Game Board): On found brick
        : Add 1 to Matched
        (Counter_ScoreADD) : Add 100 * ScoreMultiplier to Counter
```

Figure 58. Event On found brick

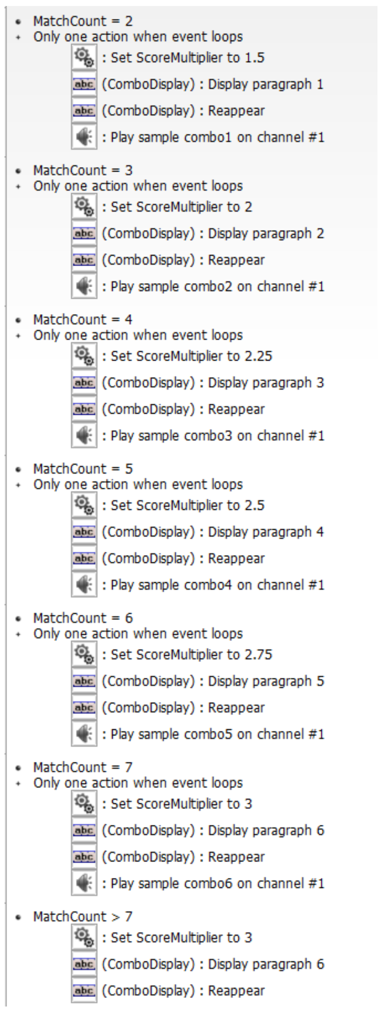
A four-of-a-kind is always worth 150 extra points and a five-of-a-kind is worth 300 points. Making two three-of-a-kind-matches at the same time gives no bonus points. (Figure 59)



```
• MatchTimer = 0
        : Set Matched to 0

• Matched = 4
+ Only one action when event loops
        (Counter_ScoreADD) : Add 150 to Counter

• Matched = 5
+ Only one action when event loops
        (Counter_ScoreADD) : Add 300 to Counter

• Matched = 7
+ Only one action when event loops
        (Counter_ScoreADD) : Add 150 to Counter

• Matched = 8
+ Only one action when event loops
        (Counter_ScoreADD) : Add 300 to Counter

• Matched = 9
+ Only one action when event loops
        (Counter_ScoreADD) : Add 450 to Counter

• Matched = 10
+ Only one action when event loops
        (Counter_ScoreADD) : Add 600 to Counter
```

Figure 59. Rewarding matches with four or five-of-a-kinds.

If MatchCount reaches more than 1, it means that there has been a combo created after the initial swap. Whenever a MatchCount threshold is reached, the ScoreMultiplier is set to the specific value, a combo sound is played and the combo displaying text object in the game is set to appear with the text signaling the size of the combo (Figure 61). The ScoreMultiplier is capped to 3, reached at the MatchCount of 7, to keep the scores at a reasonable level. (Figure 60)



```
• MatchCount = 2
• Only one action when event loops
      : Set ScoreMultiplier to 1.5
      (ComboDisplay) : Display paragraph 1
      (ComboDisplay) : Reappear
      : Play sample combo1 on channel #1
• MatchCount = 3
• Only one action when event loops
      : Set ScoreMultiplier to 2
      (ComboDisplay) : Display paragraph 2
      (ComboDisplay) : Reappear
      : Play sample combo2 on channel #1
• MatchCount = 4
• Only one action when event loops
      : Set ScoreMultiplier to 2.25
      (ComboDisplay) : Display paragraph 3
      (ComboDisplay) : Reappear
      : Play sample combo3 on channel #1
• MatchCount = 5
• Only one action when event loops
      : Set ScoreMultiplier to 2.5
      (ComboDisplay) : Display paragraph 4
      (ComboDisplay) : Reappear
      : Play sample combo4 on channel #1
• MatchCount = 6
• Only one action when event loops
      : Set ScoreMultiplier to 2.75
      (ComboDisplay) : Display paragraph 5
      (ComboDisplay) : Reappear
      : Play sample combo5 on channel #1
• MatchCount = 7
• Only one action when event loops
      : Set ScoreMultiplier to 3
      (ComboDisplay) : Display paragraph 6
      (ComboDisplay) : Reappear
      : Play sample combo6 on channel #1
• MatchCount > 7
      : Set ScoreMultiplier to 3
      (ComboDisplay) : Display paragraph 6
      (ComboDisplay) : Reappear
```

Figure 60. Increasing ScoreMultiplier and playing combo sounds.

For the system to work properly, the GameTimer had to be set to a high enough number that the combo would not be reset when the bricks were still moving, but also a low enough number so the game would not feel sluggish, as one requirement for the player selecting pieces and making swaps is that the ScoreMultiplier is 1, so the player cannot make swaps while a previous combo is running. This makes the game feel less responsive than in the prototype when no matches are being made, but the speed of the gravity in the game is set high enough that it is not a huge issue. This is a thing that could be adjusted in further development. The combo sounds are all the same sample, but the higher the combo, the higher the pitch of the sample becomes. The only other sounds in this version of the game are the whipping sound that is played whenever the player initiates a swap, and a negative sounding sample whenever a player makes an illegal swap.

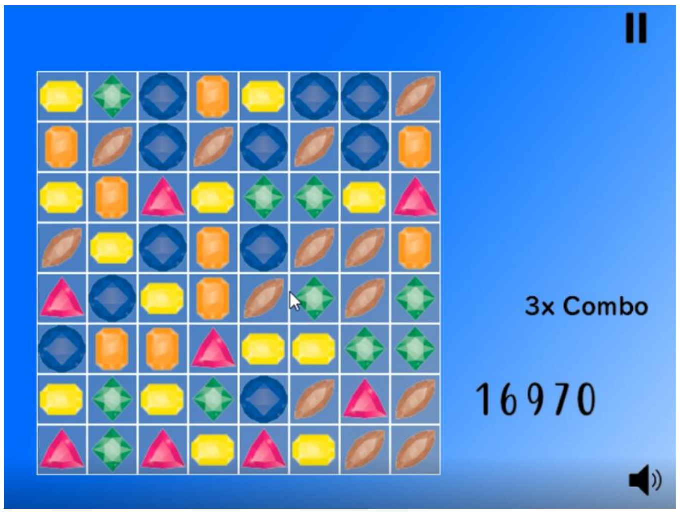
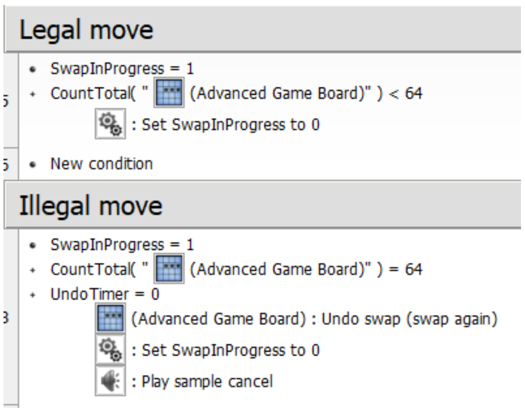


Figure 61. The score multiplier display.

5.10 Illegal move

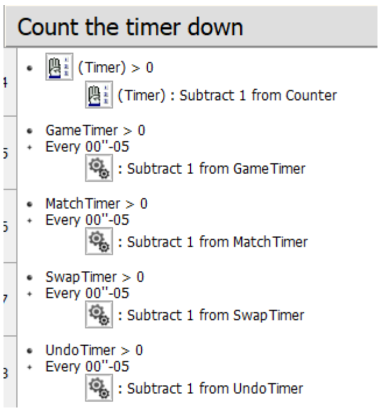
Whenever the player makes an illegal swap in this version, the swap is still performed and not outright blocked like in the prototype. After swapping, the variable UndoTimer is set to 10. If the variable reaches 0 while SwapInProgress is still 1 (as in not made 0 by the Legal move event) the pieces are returned to their original positions with a sound effect. (Figure 62)



```
Legal move
  • SwapInProgress = 1
  • CountTotal( " (Advanced Game Board)" ) < 64
        : Set SwapInProgress to 0
  • New condition
Illegal move
  • SwapInProgress = 1
  • CountTotal( " (Advanced Game Board)" ) = 64
  • UndoTimer = 0
        (Advanced Game Board) : Undo swap (swap again)
        : Set SwapInProgress to 0
        : Play sample cancel
```

Figure 62. Legal and illegal move.

All the timer variables are counted down at certain time intervals (Figure 63).



```
Count the timer down
• (Timer) > 0
      (Timer) : Subtract 1 from Counter
• GameTimer > 0
+ Every 00"-05
      : Subtract 1 from GameTimer
• MatchTimer > 0
+ Every 00"-05
      : Subtract 1 from MatchTimer
• SwapTimer > 0
+ Every 00"-05
      : Subtract 1 from SwapTimer
• UndoTimer > 0
+ Every 00"-05
      : Subtract 1 from UndoTimer
```

Figure 63. Counting down all required timers

5.11 Audiovisual design

Most of the game was graphically designed using Photoshop. Some graphics, as in mostly the gems, are freely available vector art that has been slightly edited for the purposes of the game [12]. Audio design was made using Reaper and software synthesizers, and the whip sample for swapping is a free sample found from the internet which increases the satisfaction from the very quick swapping in the game.

5.12 Afterthoughts on Fusion

Developing with Fusion is fun and there is a rather sizeable community for it, so it left a decent impression. Making menus, adding sounds and other such things are incredibly easy with Fusion, and certain kinds of games are well suited to be developed with Fusion, such as platformers. There are some interesting properties with Fusion, for example you can access Arrays out of their bounds without any issue, unlike traditional programming languages. It is a very powerful engine with not too many limits, but some of the advanced things are hard to learn and require a lot of experience with the platform, regardless of the programming style.

Certain things left something to be desired. In the event editor, it is not possible to comment out single lines, only whole events, which makes trying out small changes and experimenting a pain compared to regular programming languages. This seems odd to me. While trying to add some more advanced functionality to the game, bugs were encountered, and some of them felt like bugs in the engine itself, even though they could have also been personal mistakes, and they would have been possible to be worked around. The Sub-Application for the Pause Menu was a good idea on paper but turned out to be buggy, which was disappointing.

All in all, Fusion is a nice 2D engine for beginners and advanced developers alike with a very good amount of packed in features, but in hindsight, Unity would have probably been a better choice, especially if the aim is to make a commercial product. There are not too many popular commercial games made with Fusion, and it is kind of easy to see why it is like that.

5.13 Testing

For testing, the game was played a lot during development and also given to a few people to find bugs and playtest. This turned out to be a good method and many bugs were found and fixed during the development cycle.

6 IDEAS FOR FURTHER IMPROVEMENT OF THE GAME

The game satisfies all initial requirements, but there are still many ways in which the game could be improved.

6.1 Online leaderboards

Attempts were made to add online functionality to the game using PHP and MySQL, and ready-made PHP files for this purpose were found from the forums of Fusion. A few days were used to try to make these things work on the school server and MySQL and some other free hosts were attempted, but making Fusion connect to the database was not successful for some reason, despite all the attempts. Questions about the matter were made on the forums, and nothing helpful was answered. By making the functionality myself, it can probably be made to work but was scrapped due to time constraints. This would have definitely been easier to implement with Unity.

6.2 Animation

The game does not really have much in the way of animation at the moment, and things such as menu animations, transitions and gameplay animations could be added to liven up the game. Gameplay animations were attempted to be added, and they almost worked properly, but some very cryptic bugs appeared that led to some pieces turning invisible until matches were made, after which they would reappear. A workaround could have probably been found, but this was not a very important feature and was scrapped to make sure everything functions properly.

6.3 Improved audiovisuals

The current game has a very minimalistic style and the usage of gems as gameplay objects is not very creative, so the whole graphical style could be reworked into something more impressive and unique. Also, the game is low resolution and runs in a small window. This is not very modern, and through high resolution graphics the game could be made to run in 1920x1080, and the player could be given the selection of the window size and full screen/windowed gameplay.

Also, more audio effects could be added, and a chill music track too. I was planning to make a song for the game but was not happy with what was made, so I left the audio to just sound effects.

6.4 More game modes and settings

The genre of match-three has a huge potential for innovation and creativity. A lot of game modes and variables could be added to add more playability to the game and let the player customize the game to their playstyle.

6.5 State saving

The one “should-have” requirement that was not met was the state saving, as this was forgotten about during the development, however a quick research shows that this is doable in Fusion and could probably be added without any major changes.

7 CONCLUSION

The game was finished successfully and runs well, fulfilling most requirements set for it. Clickteam Fusion proved itself to be a strong engine, but has many negative sides such as some outdated, buggy components and the ability to not comment out single lines. However, learning the software was a valuable experience. It would be interesting to try out to convert the game to Unity and see how that would turn out.

REFERENCES

[1] Wikipedia, Unity

https://en.wikipedia.org/wiki/Unity_(game_engine)

https://en.wikipedia.org/wiki/List_of_Unity_games

[2] Steam, Clickteam Fusion

https://store.steampowered.com/app/248170/Clickteam_Fusion_25/

[3] Clickteam, Clickteam Fusion

https://www.clickteam.com/clickteam-fusion-2-5

[4] Wikipedia, Clickteam

https://en.wikipedia.org/wiki/Clickteam

[5] Wikipedia, GameMaker Studio

https://en.wikipedia.org/wiki/GameMaker_Studio

https://en.wikipedia.org/wiki/List_of_GameMaker_Studio_games

[6] Scirra, Construct

https://www.scirra.com/construct2

[7] Wikipedia, Construct

https://en.wikipedia.org/wiki/Construct_(game_engine)

[8] Raywenderlich, Unity Match 3

https://www.raywenderlich.com/673-how-to-make-a-match-3-game-in-unity

[9] ClickTeam Store, Fusion Shapes

https://www.raywenderlich.com/673-how-to-make-a-match-3-game-in-unity

[10] Cyberclic, Advanced Game Board documentation

http://cyberclic.margasoft.fr/AdvGameBoard.html

[11] Creativefabrica, Quotable font

https://www.creativefabrica.com/product/quotable/ref/87/

[12] Vecteezy, gem graphics

https://www.vecteezy.com/vector-art/108251-colorful-strass-stones

Appendix A. Research materials

Programming a Match Three Game in Multimedia Fusion

Krystian Rabe

2009