

|  |  |
| --- | --- |
|  | **American University in Bulgaria**  Department of Computer Science |

**COS 491 Senior Project I**

**Kinect Game development with Unity**

**Hristo Nenov, 100070980**

**Author:**   **, Date:**

*signature*

**Supervisor:**   **, Date:**

*signature*

**Blagoevgrad, 2013**

**Title**: Kinect Game development with Unity

**Author**: Hristo Nenov

**Abstract:** The project consists of a game developed on the Unity game engine that takes advantage of keyboard, gamepad and Xbox Kinect sensor inputs. The game is intended to be a desktop application because web applications don’t yet support the Xbox Kinect. Still a short web application version of the game that lacks the Kinect controls is also present. The game resembles the well-known brick destruction game: the user controls a paddle which is used to bounce back a ball, sending it towards bricks in the upward area. The destruction of all bricks on the current level is going to trigger the next level. Bricks drop different power-ups upon destruction with contribute to a more interesting gaming experience. There are currently ten levels inside the game. Each one with a different layout, sounds, textures, menus, all created with the help of Unity and C# scripting that has been attached to the different game objects.

**Declaration of authorship:**

“The result presented here are result of the work of the author, under the supervision of Volin Karagiozov. All sources, used in development, are cited in the text and in the Reference section.”

Author:

**Content:**

1. Introduction
2. Analysis of the application-domain problem
   1. Functional Requirements
   2. Non-functional requirements
   3. Use Cases
      1. General Game Flow
      2. Gameplay with Kinect Sensor and gamepad
3. Design of the software solution
   1. Algorithms
      1. Movement algorithm
      2. Launching ball algorithm
      3. Collision with paddle/English
      4. Others
   2. User Interface
   3. System Architecture
   4. System Decomposition
      1. Development Process details
      2. Details of installation requirements
      3. Problems encountered during development and their solutions
      4. Missing features for future development
4. Implementation
   1. Technologies
      1. Unity 3D Game Engine
      2. C#
      3. Windows Kinect SDK
   2. Approach
      1. Tutorials
      2. Bare minimal of the game
      3. More
      4. Kinect
      5. Last minute polishing
5. Conclusion
6. References

**1. Introduction:**

The project consists of a game that is developed with the help of the Unity game engine and takes advantage of the Xbox Kinect user input, keyboard and gamepad input too. The game is intended to be a desktop application because web applications don’t yet support the Xbox Kinect. Still a web application version of the game that lacks the Kinect controls is also present.

The project was developed on Unity with the help of C# scripts. It represents the standard brick game: the user controls a paddle which is used to bounce back a ball, sending it towards bricks in the upward area. The destruction of all bricks on the current level is going to trigger the next level. Bricks drop different power-ups upon destruction with contribute to a more interesting gaming experience. There are currently ten levels inside the game. Each level has a different layout, different brick types, and a background which contributes to game theme. Sounds are also present in the game. On collision with game objects, the ball is going to trigger sound effects: on collision with the paddle, a metal clinging sound is going to trigger; on collision with the wooden walls surrounding the game scene, a wood tap is going to trigger; on collision with bricks: a brick tap-like sound is going to be heard. Sound effects are also added to the dropping power-ups which contribute to the overall sound quality of the project. Background music is also going to be present throughout the game. It consists of a mixture of arcade game soundtracks from the 1980’s.

The project takes input from different sources. Keyboard input is present: movement of the paddle, pause option, quit option and the option to launch a new ball are all included. Furthermore, a gamepad input is also integrated. The same key instructions have also been implemented here: movement of the paddle, pause, quit, launch. A remark to be done here is that the control of the paddle is much swifter that the one with the keyboard because of the arcade-like type of the game. Furthermore, I succeeded into connecting the Xbox Kinect sensor with my game. The game takes input from the camera using its depth view technology and senses the movements of the user. I have attached the control of the paddle with the movement of the left hand of the current player. Movement of the paddle is restricted in the sense that it can only move in the x axis. The movement of the hand is detected and transferred to the paddle with a slight modification: I have modified the speed of the movements so the user doesn’t have to go out of his current position in order to achieve full coverage of the game scene area. A combination of the Kinect sensor and the gamepad is required if a user desires to use the Kinect. This is such because the gamepad still offers you the ability to pause and quit, which are absent with the Kinect.

During the development of the game I conducted some alpha and beta testing too, trying to follow the usual game release techniques. Alpha testing was done by me and contributed to testing the algorithms I put behind the game. Beta testing was done by people who are familiar with computer games and some that are not. Their feedback was used for improving the algorithms, controls and visuals of the final product.

I did this project because of my love for computer games. I started with the idea to teach myself how to create games with unity and then I decided to try and use the Kinect sensor to improve the overall quality and experience of the game. Doing so helped me improve a lot of skills. The main one which is the techniques used in most of the games. There are certain techniques for implementing sounds, effects, menus and a lot more items that are all essential for computer games. Creating a game on my own helped me get a better understanding of them. Furthermore, I know for fact that there hasn’t been a similar project at the American University in Bulgaria. Creating something new and fresh, something that students at AUBG might be interested in but don’t have the chance of witnessing in classes was also one of the reasons I decided to go with this project.

**2. Analysis of the application-domain problem**

* 1. **Functional requirements:**
     1. **Presentation**:

One of the functional requirements is the presentation of the game. The user has the option to choose from different resolutions and window sizes. The game is made in such way that the graphics, images and everything scales with the window size. No pixilation is going to be present even if the user has chosen the lowest resolution possible.

* + 1. **Main Menu:**

When the game is launched a main menu screen is shown. There are two options that a user can choose from: New Game and Quit Game. Choosing New Game will trigger the first level of the game. Choosing Quit Game will make the application quit. While being in a level, the user can Pause the game and then UnPause it. The user has the option to quit the game too. There are secondary functional requirements though.

* + 1. **Paddle Control**:

The main one is the control that the user takes. The user is given the control of the paddle. Using a keyboard, the user can force the paddle to move left or right using the WSAD or arrow keys. This can also be done through a gamepad. The control through the gamepad is significantly smoother. A third option is also present: the user has the option to connect an Xbox Kicent Sensor to his computer and use his left hand to control the paddle.

* + 1. **Ball Behavior:**

Another functional requirement is the control of the ball. Generally the control of the ball greatly depends on the gravity in the game, but initially the player has the option to launch it in a certain direction. This option is again given upon losing a life.

* + 1. **Destruction of bricks:**

Destruction of bricks is also an important functional requirement. The destruction of all destructible bricks is going to trigger the next level. Upon death, bricks might drop special power-ups that are going to either spawn an additional ball or contribute greatly to the players score.

* + 1. **Score:**

Score is kept throughout the game. Score is increased upon destructing different bricks. Depending on the type of the bricks, the player is going to receive different amount of points.

* + 1. **Level Counter:**

Level counter is also implemented. It shows the player the name of the current game.

* + 1. **Lives Counter:**

Lives counter can be seen in the lower right corner of the game screen. The player initially starts with three lives that can be lost upon losing all of the balls on the screen. When this counter reaches zero, the end game screen is triggered.

* 1. **Non-functional requirements:**
     1. **FPS:**

The FPS kept while in game has to be satisfying. Drops shouldn’t be noticed no matter of the input decide used by the user. Resolution and windows size should make a difference either.

* + 1. **Bugs:**

Bugs are present but should be game breaking. While some of the game objects that are affected by physics and collisions might act weird in some cases, these should make the game unplayable. They are presents because of the calculations behind the game. Some of them are very complicated and as an unexperienced game developer, I might have made some mistakes.

* + 1. **Launch time:**

Launch time of the system has to be less than 8 seconds. It takes some time for the system to check the input devices and whether a keyboard, gamepad or a Kinect sensor is available. In terms of computing power, the launch time from a system to system should be very different.

* + 1. **Gamepads support:**

The user has to be able to use a variety of gamepads with the systems. The main requirement is that the system recognizes the device. The same is for the Kinect too.

* + 1. **Deployment:**

Deployment of the game is also easy. Simple copying works the best. There is a single requirement for the system. It has to be a windows machine. While I could have compiled and created a game of Mac, I decided to keep it windows oriented.

* + 1. **Documentation and hints:**

Documentation of the project is not present in the project itself. In my desire to make it 1990’s like, I decided to keep it with less documentation and explanations about the controls, power-ups, bricks types and etc.

* 1. **Use Cases:**

**Name of use case:** General Game Flow

**Entry Condition:** The use case stars user launches the game

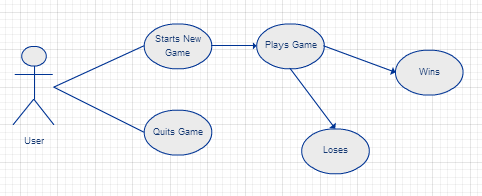
**Flow of Events:**

1. User clicks Start game and a game is started
2. User plays the game
3. User either wins or loses
4. Application exits to desktop after a message has been shown to the user

**Exit Condition:** The user has tried the game and has either ended up at a win screen, gameover screen, or none of them (back to desktop after exiting the game)

**Exceptions:**

* The user can always decide to stop playing an just enjoy the music in the background
* The user can always decide to quit the game
* A user can always enter a PAUSE state



*Figure 1. General Game flow*



**Name of use case:** Gameplay with Kinect sensor and gamepad

**Entry Condition:** The user decided to start a new game

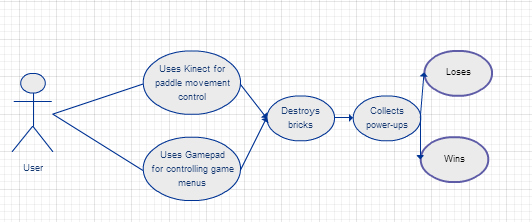
**Flow of Events:**

1. User controls the paddle through the Kinect sensor and the menus through the gamepad
2. User uses the ball to destroy the bricks on the screen
3. User collects power-ups that are dropped on a random principle upon destruction of the bricks
4. User either wins or loses depending on his skill

**Exit Condition:** The User either sees a win screen upon completing the 10 levels, or losses because he has lost his 3 lives before completing the complete game

**Exceptions:**

* The user can always decide to stop playing an just enjoy the music in the background
* The user can always decide to quit the game
* A user can always enter a PAUSE state



*Figure 2. Gameplay with Kinect Sensor and Gamepad*

1. **Design of the software solution**
   1. **Algorithms**
      1. **Movement Algorithm**

The algorithm used for calculating the movement of the paddle is made in such a way that it works both with input from the keyboard and input from the gamepad. A few optimizations where made in order to achieve this result. The heart of the algorithm is the way movement is detected. Instead of detecting key strokes on the keyboard or the gamepad, I decided to detect movement on the x axis which includes both input from the WSAD keys, the arrow keys, and the gamepad controls. The movement on the axis is taken and multiplied by a constant. This constant determines the speed of the paddle. It can always be changed by a developer. The result is also multiplied by another variable: Time.deltaTime. The explanation behind this multiplication is as follows:

This algorithm is called in the Update() function. This function is called every frame the game is ran. On some machines, the game is going to produce more frames than on others. This means that the function is going to be called more times and this will lead to the paddle movement varying from a machine to a machine. This is why I have decided to multiply the speed of the paddle by a constant which represents a period of time past in real life. The result of this will be: the speed is going to be the same on every machine, because it would depend on real time, not on frames past in the game.

This result of the multiplication of the speed taken from the axis, the speed constant and the time constant is then applied to the paddle.

* + 1. **Launching ball**

Whenever a ball has to be launched, the positing of the paddle is taken, so the ball can be placed in the middle of the paddle. The Ball is placed in the middle of the paddle and slightly in the air. This is done in order to avoid bugs, where the ball would get stuck to the paddle. The Ball’s movement is disabled while on the paddle so it moves with the movement of the paddle. Upon launch, the movement of the ball is enabled and physics, gravity including, start applying to the ball. Next, a force is applied to the ball. Two parameters are very important here:

* Speed of the ball: this is defined by the developer and it is a constant which is to be kept throughout the whole game. This is done in order to keep the ball from reaching game-breaking speed, or a speed too slow for the game.
* Speed of the paddle: the speed of the paddle is taken, multiplied by the speed of the ball and applied to the ball. The reason behind this is that it gives the user a chance to send a ball in a direction he wants. If the user decided to launch the ball while moving right, the ball will start flying in the same direction.
  + 1. **Collision with paddle / English**

Whenever a ball collides with the paddle, this algorithm is executed. The main reason behind it is to apply real life physics to the ball movement. Upon collision, the contact points of where the ball has bounced are saved in a variable. These contact points consist of point in both the paddle and the ball. Depending on where the contacts are, forces are added to the ball, or the so called English is applied:

If the ball hits the paddle near the end of the paddle, a stronger force is added that might change the direction of the ball

If the ball hits the paddle somewhere around the middle, the force is going to be much smaller, but still existent.

The reason behind this algorithm is so that the user has more control over the ball and that he can send it wherever he wants to, as long as he is good with predicting the place of collision

* + 1. **Others**

Other small algorithms are also present. Algorithms that are responsible for the behavior of the power-ups, lives, and score are also implemented but they are not complex enough to be explained additionally. Simple calculations are present in their implementations but necessary for the execution of the game

* 1.  **User Interface**

*Figure 3. Main Menu*

* + 1. Main Menu

The first part of the interface is the main menu. The user has the options to start a new game or quit the game. Input can be taken from mouse, keyboard and gamepad. Background music is played while waiting for user input.

Animations are present on this screen. Upon mouse overing an option, an animation is going to be triggered enlarging the option. Upon removing the mouse from the opting, and animation that returns the letters to their normal size is going to be triggered.

These animations are recorded in advanced and are only triggered on calling them on certain events

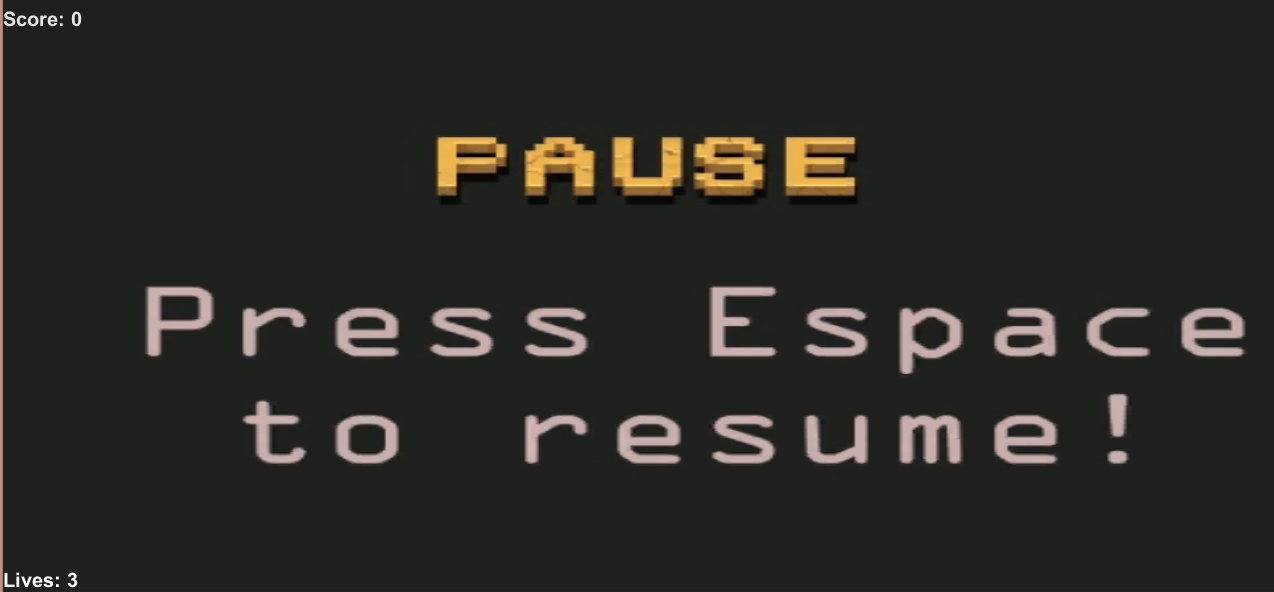


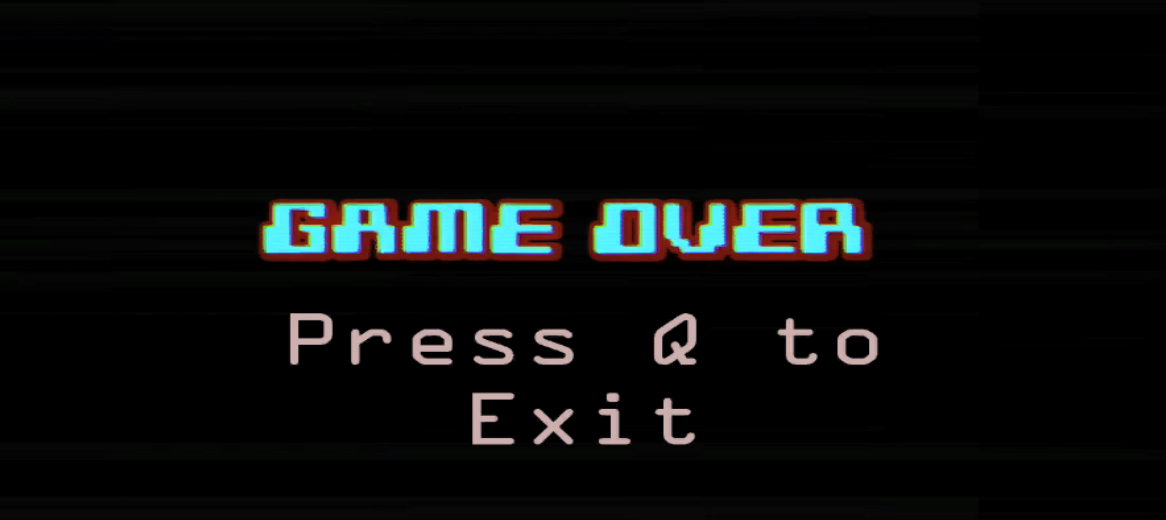
*Figure 4. Main Interface*

* + 1. Main Interface

The main user interface combines a couple of items:

1. Score counter
   * Keeps the score achieved by the player
2. Level counter
   * Keeps count of the current level
3. Lives counter
   * Keeps track of the remaining lives of the player
4. Game level
   * Here is the heart of the game. It holds the level layout, balls, paddle, bricks, physics, etc
5. Kinect screens
   * The two screens on the right contain the input taken from the Kinect. While the first one contains the image taken from the Kinect camera, the second one contains the DepthView image taken from the Kinect camera, which shows the distance between objects in the camera.

*Figure 5. Pause Screen*



*Figure 6. Game Over Screen*

* + 1. Pause, Win and Game over screens

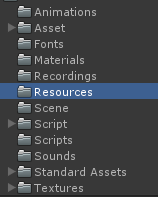
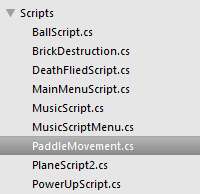
These screens are triggered on specific ocasions and can be exited with the right commands shown on the screens

* 1. **System Architecture**

The system architecture is made out game objects interacting which each other with the help of mechanics implemented in Unity or through C# script.

Different scripts have been attached to different object so that specific behavior has been given to them. Each script contains functions that are called on different occasions. Some scripts hold one or two functions, while others that are ran throughout the whole game, contain more important functions. Communication between the different script is always present.

Apart from that, sounds, fonts, animations, textures, materials have also been applied to the game objects so that the over quality gets improved.

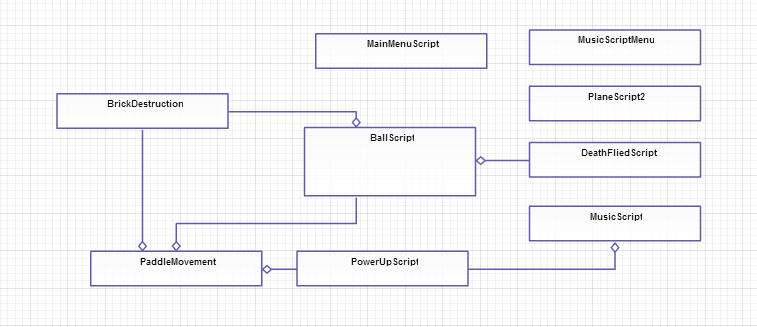


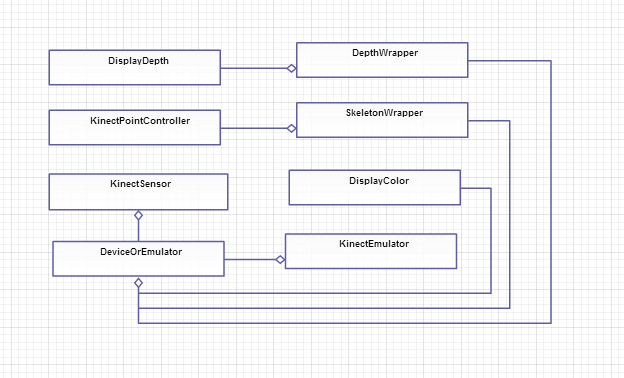
*Figure 7 & 8. Main Scripts and Game Architecture*

Furthermore, we have Kinect scripts taken from the Microsoft Kinect SDK for Windows that I have modified so that they are applicable to my projects.

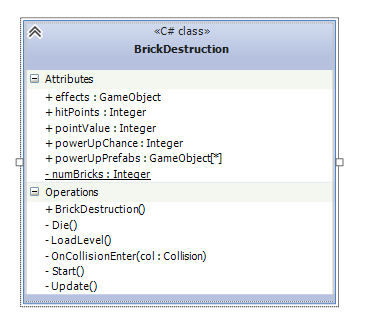
* 1. **System Decomposition**

The number of scripts created entirely by me is nine. The main ones are BallScript and PaddleMovements. This is such because the paddle movement script is attached to the paddle with is never destroyed in game. This makes the script perfect for holding importation information such as Lives and Score. The BallScript is important because the game is concentrated around controlling the ball.

*Figure 9. Connections between main scripts*

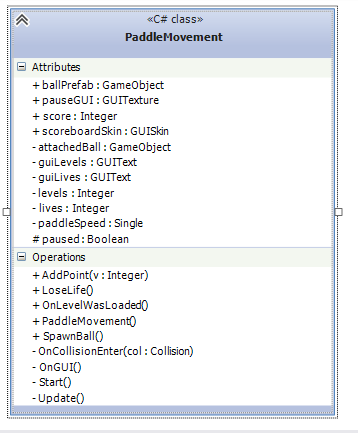


*Figure 10. Connections between Kinect scripts*

*Figure 11. BrickDestruction UML class diagram*

* + 1. **BrickDestruction**

This class is responsible of the behavior of the bricks. Is takes care of changes the bricks undergo, the destruction of the bricks themselves and the appearance of the powerups. This class is attached to every brick in the game, and contains a public variable that determines the possibility of a powerup to be dropped. It also holds a variable that is responsible for the life points of the bricks. This class is also responsible for the loading of levels upon destruction of all bricks on the screen.



*Figure 12. PaddleMovementScript UML class diagram*

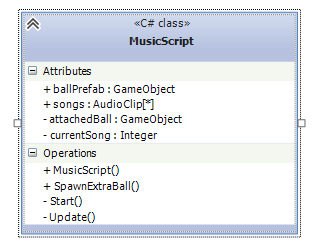
* + 1. **PaddleMovementScript**

The PaddleMovementScript is attached to the paddle. This script takes care of the control of the paddle. In addition, it also takes care of keeping the score of the player, his lives and the number of the current level. The rest of the controls present in the game are also included here: pausing and quitting the game.

*Figure 13. BallScript UML class diagram*

* + 1. **BallScript**

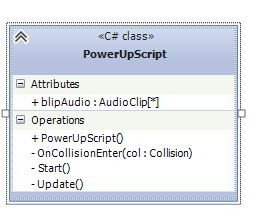
The BallScript class is attached to every ball that appears on the screen. At includes the control of the sounds that are played upon collisions. It also contains come instructions that influence the movement of the ball



*Figure 14. Music Script UML class diagram*

* + 1. **MusicScript**

The MusicScript class is attached to an invisible game object, and thus turning it into a speaker. It contains an array of songs that are filled by the developer. During the execution of the game, it takes care of playing the playlist. Another function called SpawnExtraBall is written in this class. This function takes care for the creation of an extra ball upon collecting the pink power-up.

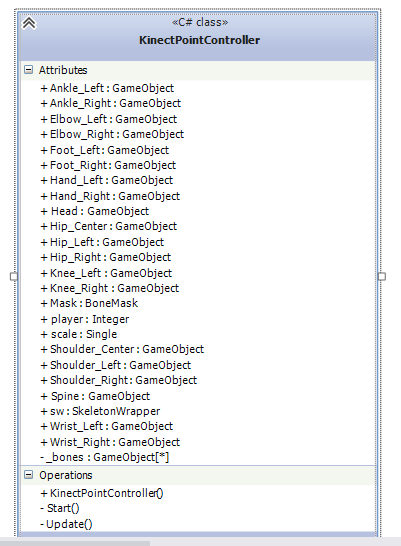


*Figure 15. PowerUpScript UML class diagram*

* + 1. **PowerUpScript**

The PowerUpScript is attached to the power-ups that get randomly created. Depending on the type of the power-up, this class does two thinkgs. First, it destroys itself on collision with the paddle in order to create the effect of absorption. Second, it:

* Either adds additional points to the player’s score if the blue power-up is picked
* Or spawns an extra ball to be controlled by the player if the pink power-up is picked. The core of this function is in another class in order to avoid some bugs and game breaking mechanics.



*Figure 16. KinectPointController UML class diagram*

* + 1. **KinectPointController**

This class is taken from the windows Kinect SDK and has been modified by me. It does the connection between the skeleton of the user detected by the Kinect sensor and the object that is connected to the skeleton. In our case that is the paddle. It contains a lot of gameobjects that can intended to be attached to the different parts of the user’s body.

* 1. **Development Process details**
     1. **Details of installation requirements**

The installation requirements for the project depend on the way the user wants to use it. In order to take full advantage of the system, the player should get a hold of a gamepad and an Xbox Kinect Sensor. Drivers for both are needed. I suggest using the Microsoft official drivers for the Kinect Sensor, because I used the official SDK.

* + 1. **Problems encountered during development and their solutions**

I encountered a couple of problems during the development of this system:

* Having a completely different idea for the project.
  + At first I wanted to do a web site written on RubyonRails with which help users would have the chance to write Cucumber tests easily. Later in the semester I realized that this project wasn’t giving me enough satisfaction. Halfway through the semester I decided to develop a game on Unity instead. After I contacted my adviser, I completely switched my focus from RubyOnRais to Unity
* Getting used to Unity
  + Because of the absence of any experience with the Unity engine, I experienced great difficulties getting used to it. I was completely new to the idea of game development, and I failed to understand basic mechanics and techniques on several occasions. It took me several tries to even start the projects, but in a couple of days, I was already having results and saving progress.
* Connecting the Kinect
  + Connecting the Kinect Sensor to my computer and then to the Unity game engine turned out to be quite the difficult tasks. There are a lot of drivers out there and it took me a couple of days the find the right ones. Each driver was intended to be used with different algorithms and techniques for implementing solutions. I found out that the official Microsoft SDK served well for the purpose of my project.
    1. **Missing features for future development**

There are several features that I wanted to implement but didn’t have the time to.

* The first one is a high score database. I wanted to create a simple database that would keep track of the highest scores achieved by players
* Full control with the Kinect. I initially wanted to implement full control supported by the Kinect. The difficulties I faced connecting it to my solutions took too much time and effort resulting in me changing some aspects of the game
* Improved overall quality of the game. There are a lot of things that can be improved in the game: graphics, physics, materials, level designs.

1. **Implementation**
   1. **Technologies**
      1. **Unity 3D Game Engine**



*Figure 17. Unity logo and program layout*

Implementation was done using the Unity Game Engine. The Unity Game Engine gives the developer the ability to create game on different platforms. The developer has the ability to create objects, modify physics, create effects, design levels and in general everything that is needed for the creation of a game. Unity also gives the developer the opportunity to develop a product for different systems: android, flash, windows, mac, etc.

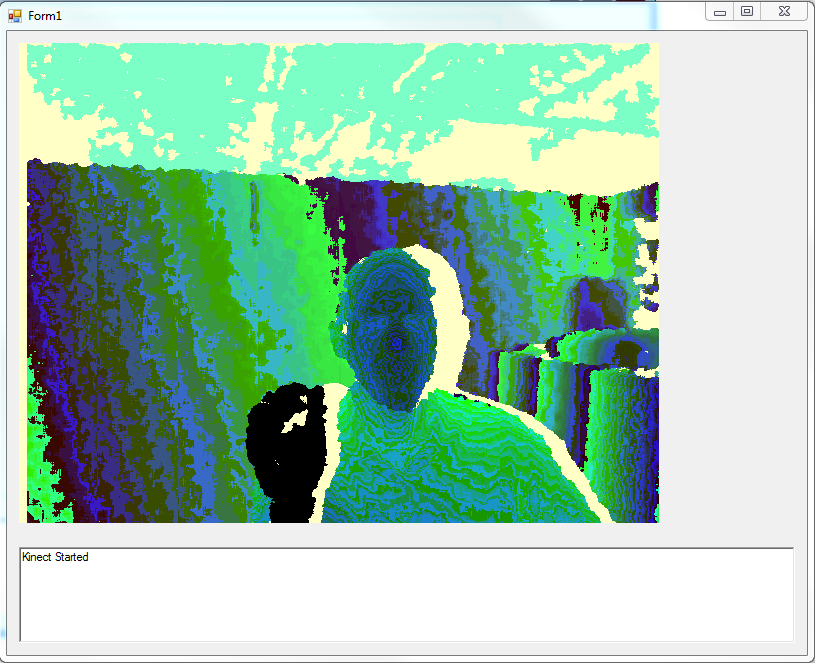
* Rendering
  + The graphics engine uses Direct3D, OpenGL, and OpenGL ES depending on the platform. There is support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion, dynamics shadows using shadow maps, render-to-texture and full-screen post-processing effects. Unity also supports art assents and file formats from 3ds, Max, Maya and many more. I took advantage of this, and created some materials in Photoshop
* Scripting
  + The game engine’s scripting is built on Mono 2.6, the open-source implementation of the .Net Framework. Developers can use UnityScript, C# or Boo.
* Asset tracking
  + Unity also includes the Unity Asset Server - a version control solution for the developer's game assets and scripts. It uses PostgreSQL as a backend, an audio system built on the FMOD library (with ability to playback Ogg Vorbis compressed audio), video playback using the Theora codec, a terrain and vegetation engine, built-in lightmapping and global illumination with Beast, multiplayer networking using RakNet, and built-in pathfinding navigation meshes.
* Platforms
  + Unity supports deployment to multiple platforms. Within a project, developers have control over delivery to mobile devices, web browsers, desktops, and consoles. Unity also allows specification of texture compression and resolution settings for each platform the game supports.
  + Currently supported platforms include Xbox One, BlackBerry 10, Windows 8, Windows Phone 8, Windows, Mac, Linux, Android, [iOS](http://en.wikipedia.org/wiki/IOS" \o "IOS), Unity Web Player, Adobe Flash, PlayStation 3, Xbox 360,Wii U and Wii. Although not officially confirmed, Unity also supports the PlayStation Vita as can be seen on the games Escape Plan and Oddworld: New 'n' Tasty.
* Physics
  + Unity also has built-in support for Nvidia's (formerly Ageia's) PhysX physics engine (as of Unity 3.0) with added support for real-time cloth simulation on arbitrary and skinned meshes, thick ray casts, and collision layers.
    1. **C#**

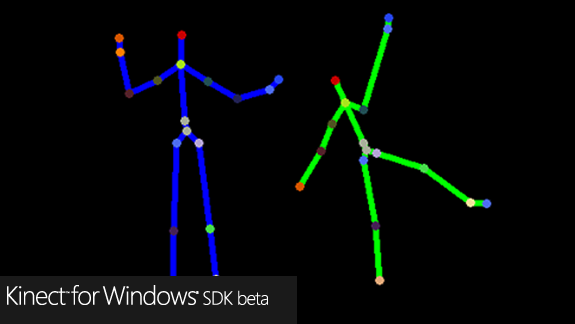
While developing in Unity, the developer has to write scripts in order to attach them to game objects and give them functions and characteristics. While the engine gives you the choice of writing in JavaScript too, I chose to do it in C#. I did so because I feel more familiar with the C# scripts and decided to improve my skills in that particular language

* + 1. **Windows Kinect SDK**

The SDK provides Kinect capabilities to developers to build applications. I chose to you these one, because I found the documentation easier to understand that others. The SDK includes the following features:

* Raw sensor streams: Access to low-level streams from the depth sensor, color camera sensor, and four-element microphone array.
* Skeletal tracking: The capability to track the skeleton image of one or two people moving within the Kinect field of view for gesture-driven applications.
* Advanced audio capabilities: Audio processing capabilities include sophisticated acoustic noise suppression and echo cancellation, beam formation to identify the current sound source, and integration with the Windows speech recognition API
* Sample code and Documentation.

*Figure 18. One of Kinect’s abilities: Depth View*



*Figure 19. One of Kinect’s abilities: skeleton tracker*

* 1. **Approach**

The approach I took was very secure. I was keeping a few versions of the game at every point. Whenever I had a feature ready, I saved the project and created a backup. At the end of a series of successful changes, I even backup a copy online in case something had happened with my computer

* + 1. **Tutorials**

I started watching tutorials on YouTube on developing games with unity. I tried out some of them to be sure I was learning and remembering. Though the engine is not the newest one, there are not that many good tutorials. I even had to watch some streaming from programming competitions so I can get a hold of more knowledge.

* + 1. **Bare minimal of the game**

I developed the bare minimal of my game. It had only one level, no GUI, no power-ups. What it had was the physics needed and some objects so the player can “play” some of it.

* + 1. **More**

I started improving everything I had. I created more levels, types of bricks, added sounds. In general, I improved the overall quality of the gaming experience.

* + 1. **Kinect**

I added the Kinect sensor. After having the game almost finished, I tried adding the Kinect. After a few unsuccessful tries, it finally worked.

* + 1. **Last minute polishing**

Last couple of days I spent on fixing bugs, improving code, adjusting game mechanics and here and there some cosmetic changes

1. **Conclusion**

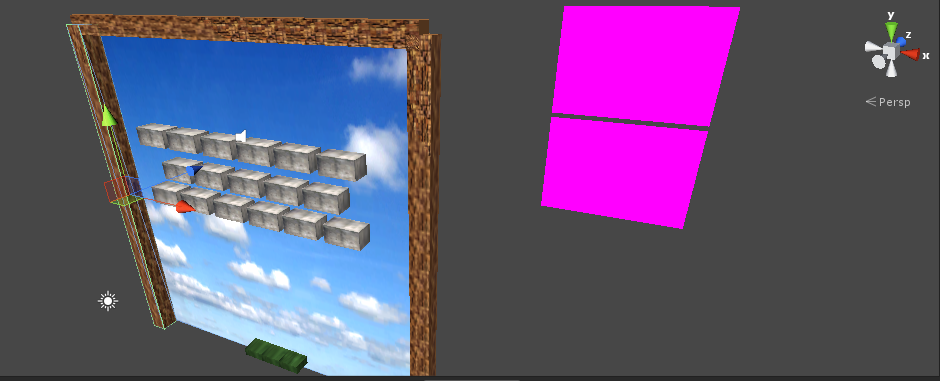
The results of the project were very pleasing. At the end, I got a full working game that supports input from 4 different devices: mouse, keyboard, gamepad, and Kinect sensor. It has 10 different levels, engaging mechanics, and beautiful music effects. I started it with low expectations, waiting for something to crash and stop me from creating something working but I managed to complete most of the goals I had set up for myself.

There are a couple of features that I wasn’t able to implement, but I am planning on spending some more time on this project. One of the them is a database that keeps track of users’ highest scores and nicknames. Another feature that I tried to implement on a several occasions but I didn’t have success is attaching an online radio to the program. The idea was to use endless radio stream attached to the “speaker” in the game so that the package of the game is smaller. Currently, the music tracks in the game take around 15mb. Attaching an online radio stream to the game would save some space from the package and the game would because easier to distribute.

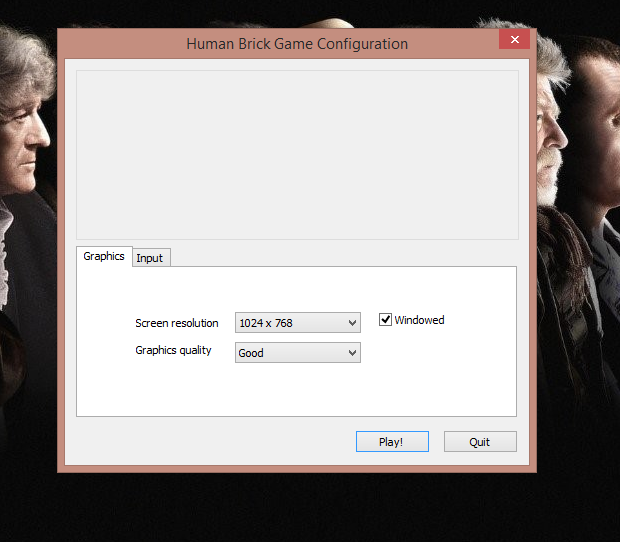
Another feature that I couldn’t implement is full Kinect support. I managed to research some worked and had little success with recording gestures made with hands and then attaching them to actions specific to the game. The results though weren’t concrete and cause the game to crash on several occasions. That is why I decided to not include them in the final release.

Working on this project, I managed to inherit a lot of skills in areas that I haven’t explored. I succeeded in learning how to use Unity 3D Game Engine in its basics and some more advanced areas. I also learned how to use the Kinect SDK in order to connect it to other pieces of software. Improvement in my C# skills is also present. Encounters of problems during the development of this project were tripping me and my progress from time to time. They were mostly cause because of my lack of experience with the platform and technologies.



*Figure 20. Main Menu with animated options*

*Figure 21. Game’s universe in 3D*



*Figure 22. Options for screen resolution and input provided by Unity*

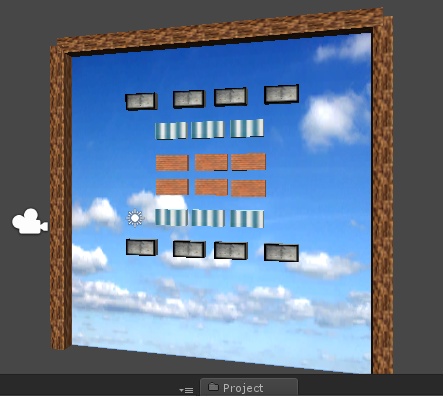
*Figure 23. Level 1*

*Figure 24. Level 2*





*Figure 25. A level with several balls in play at the same moment*



*Figure 26. Layout for level 10 including 3 different brick types*

1. **References:**

"Microsoft Kinect-Microsoft SDK." *Unity3D RSS*. N.p., 5 Sept. 2011. Web. 01 Dec. 2013.

"Start Creating Today Pay as You Go." *Unity*. N.p., n.d. Web. 01 Dec. 2013.

"Tower Dive!" *Tower Dive*. N.p., n.d. Web. 01 Dec. 2013.

"Unity - Learn - Documentation." *Unity - Learn - Documentation*. N.p., n.d. Web. 01 Dec. 2013.