# Introduction

My problem is to design and produce a prototype for a functional and entertaining 2D platforming game that my user would want to purchase. My user would typically be teenagers interested in video games but could extend to anyone who would be interested enough to buy my game. Specifically, I will be referring to Owen Barker, a 17-year-old who regularly plays video games, in order to get feedback during development.

# Identifying the problem

**3.1.1 Problem identification**

**(a) Describe and justify the features that make the problem solvable by computational methods.**

There are multiple comparable products currently on the market, such as Dustforce and N++, all of which share similar features. Below is a list of common features and why these leant themselves to a computational solution:

* Physics based character movement, with either real world physics or a custom physics engine for the game. This would involve numerous calculations every second, and would need to be consistently accurate, making it impractical to perform without the use of a computer.
* An in-built level creator, allowing users to create and play their own levels. These levels can be stored locally, allowing players access to them when required. The files could also be stored on the cloud for the player, as a backup copy, or to allow the user to access them from computers other than the one the level was created on. As a digital file, the levels would be easy to manage and edit as required.
* Global level sharing of user created levels, through the use of user content distribution services such as Steam’s Workshop, or through a system set up specifically for the product. This would require access to a server and would have to be connected though the internet to achieve the global scale.
* A scoring system for the levels, based on the time taken to complete the level, or the number of points accrued through the level. The ability for this score to be stored allows the player to try and beat their highest scores and could also be used to show the user statistics about how they play, such as a ranking and average percentage score for the game. It would be unrealistic to perform all those calculations by hand every time a statistic changed, but a computer would find the task trivial.
* Global leader boards for level scores, either through an online gaming network’s API, or through a system specifically set up to accommodate the service. Either requires access to a server with a database of levels and the scores of every player. This would likely be hundreds of thousands of pieces of data, making it impractical to handle without computational methods.
* Multiplayer with other people on the same machine. This would directly increase the number of calculations being performed each frame, reinforcing the need for a computed physics simulation.
* Replays of both standard and custom levels, allowing the user to watch their previous successful attempts and view the replays of the global best at that level. A lot of information would need to be kept track of every time a level was attempted, but then quickly disposed of if the attempt was unsuccessful. This data management could be easily performed by a computer, along with saving the replay on the successful attempts.
* A large quantity of varying levels which the user progresses through either linearly or non-linearly. Each of these levels will require data stored about them, and the players progress through them will need to be tracked. This lends itself towards a computational solution because all of that data can easily be handled by a computer.
* The ability to choose the input to the system, for those versions not restricted to the hardware of a games console. From a functional and common keyboard and mouse, to an ergonomically designed gaming controller, this choice is only available because the computer can support the change in input devices.
* A range of audio and visual assets are used throughout the game, conveying what is occurring to the user in better detail than text alone could. They also add appeal to the product, giving it a memorable style. Some of these assets are used on cues calculated by the physics engine, whilst others would be part of the background effects. This means that, for audio assets, two sounds could need to be played simultaneously, whilst multiple visual assets are always going to be required on screen at once. A computer would be able to handle these concurrent processes much better than any other method.

**(b) Explain why the problem is amenable to a computational approach.**

I have therefore decided to computerise my solution for the following reasons:

* As shown above, similar systems currently in place show that the problem lends itself towards a computational solution.
* The overall problem can be broken down into smaller problems, such as character movement and enemy AI. These smaller problems would be easier to solve than the whole solution at once, increasing the likelihood of my solution being successfully completed in time.
* Sections of the solution can be reused in various places. For example, a level creator could be created, and then used to create the content for the solution.
* My problem will contain multiple mathematical calculations per second, such that it would be impossible to solve in any practical way other than through the use of a computer. Isolated calculations could be processed concurrently by taking advantage of parallel processing (so long as the user’s machine has a multi-core CPU)
* There will be a regular need to process large quantities of data, either scores or game statistics. This can be achieved quickly and efficiently by a computer.
* Some aspects of my solution could require accessing data from users across the globe. The easiest way to achieve this is through the data being on a computer that is connected to the internet.

# Stakeholders

**3.1.2 Stakeholders**

**(a) Identify and describe those who will have an interest in the solution explaining how the solution is appropriate to their needs (this may be named individuals, groups or persona that describes the target end user).**

The two primary stakeholders are myself and the potential users of my game, because both seek to benefit from my solution. I am going to use my friends as a selection of the potential users of my game, in order to get their feedback on my solution, and to support whole system testing later on in my project development. I could have a financial interest in the success of my product, should I take it to market, and will stand to gain experience in software and game development through the creation of this solution. I will also likely end up using my product myself, so will share the needs of the users.

The users will want the game to be fun and engaging, and worth their money should I take it to market. They would want to know that the software is safe to install on their machines, and for the software to be as compact as possible, to minimise the impact on storage space.

They would expect a functional product.

My solution will help by providing a functional game to the users, that is fun for them to play. The process of making my solution will provide me with experience in software and game development, and when the solution is finalised, possibly after the initial version created for this coursework, I may decide to put my product on the market.

# Research of Existing Systems

**3.1.3 Research the problem**

**(a) Research the problem and solutions to similar problems to identify and justify suitable approaches to a solution.**

In order to better understand my problem and be able to make justifiable decisions when designing the solution, I decided to research some of the current systems that solve a similar problem. This research ranged from analysis of the system itself, to researching the development of the solutions and identifying issues that I may face. This will allow me to learn from the experience of others, and plan around potential issues by taking them into account beforehand.

# System 1: Dustforce

Development studio: Hitbox Team

Release price: $10

Platforms: Initially Windows, expanded to Mac and Linux

Minimum system requirements:

* Core Duo 2.0 GHz or equivalent
* 1 GB RAM
* Dedicated graphics card
* Shader Model 2.0 support
* 400Mb available storage space

Development tools: Custom game engine, prototype made with GameMaker Studio

Estimated production cost: ~$140k

Dustforce started out as a prototype being submitted to a games contest. 2 people worked on the prototype for 4 months, finishing in time for the contest deadline. After winning the contest, the full 4-person development team focused on bringing the prototype to a full release over the course of 1.5 years. The prototype made in GameMaker Studio was ported to their own C++ game engine during this process.

Dustforce is a 2D platformer where the aim is to flow through each level, clearing dirt and grime from the environment. The player can jump and double jump to reach areas, combining wall running and a dash to achieve a masterful flow through the level. Clearing all the dirt in a level, and maintaining a flow whilst doing so, will grant you keys to unlock harder levels, and progress through the game.

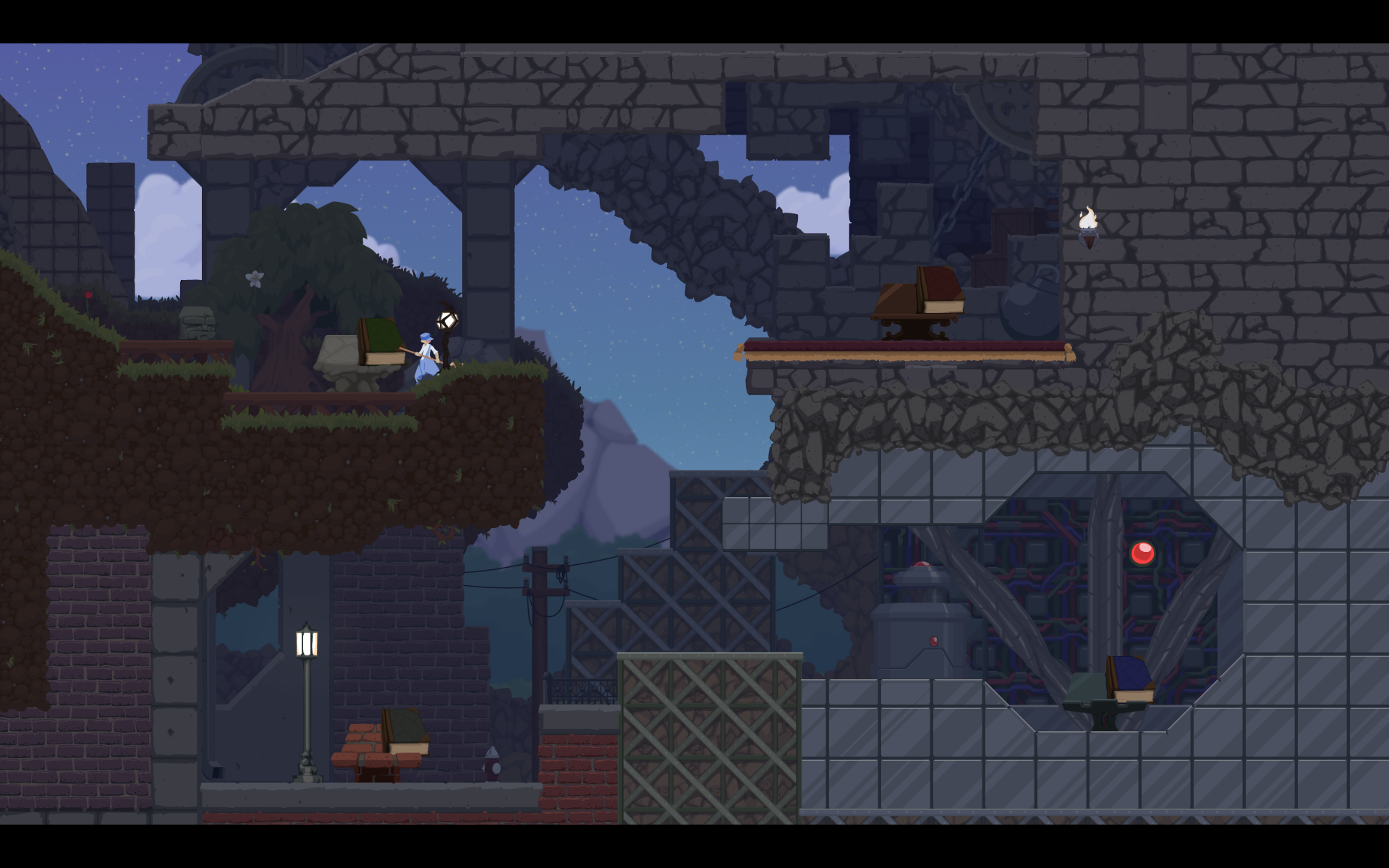


Figure 1



Figure 2

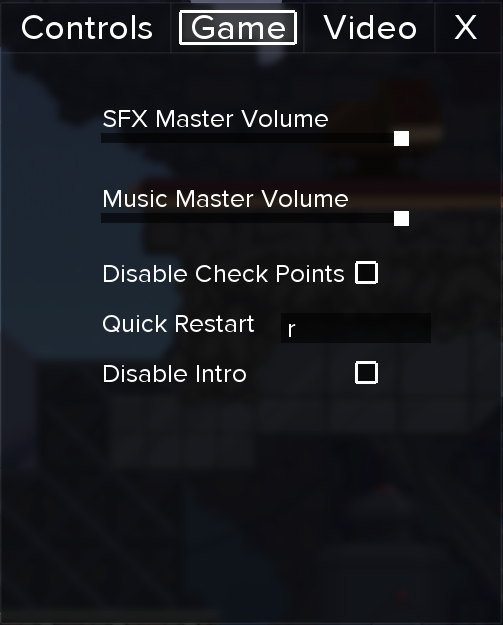
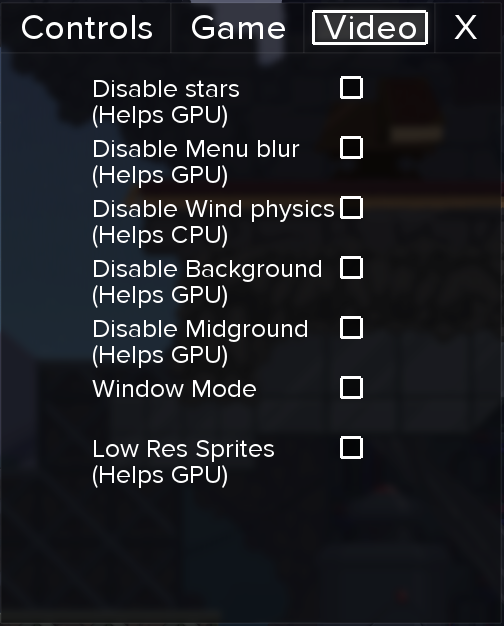


Figure 3 Figure 4



Figure 5



Figure 6

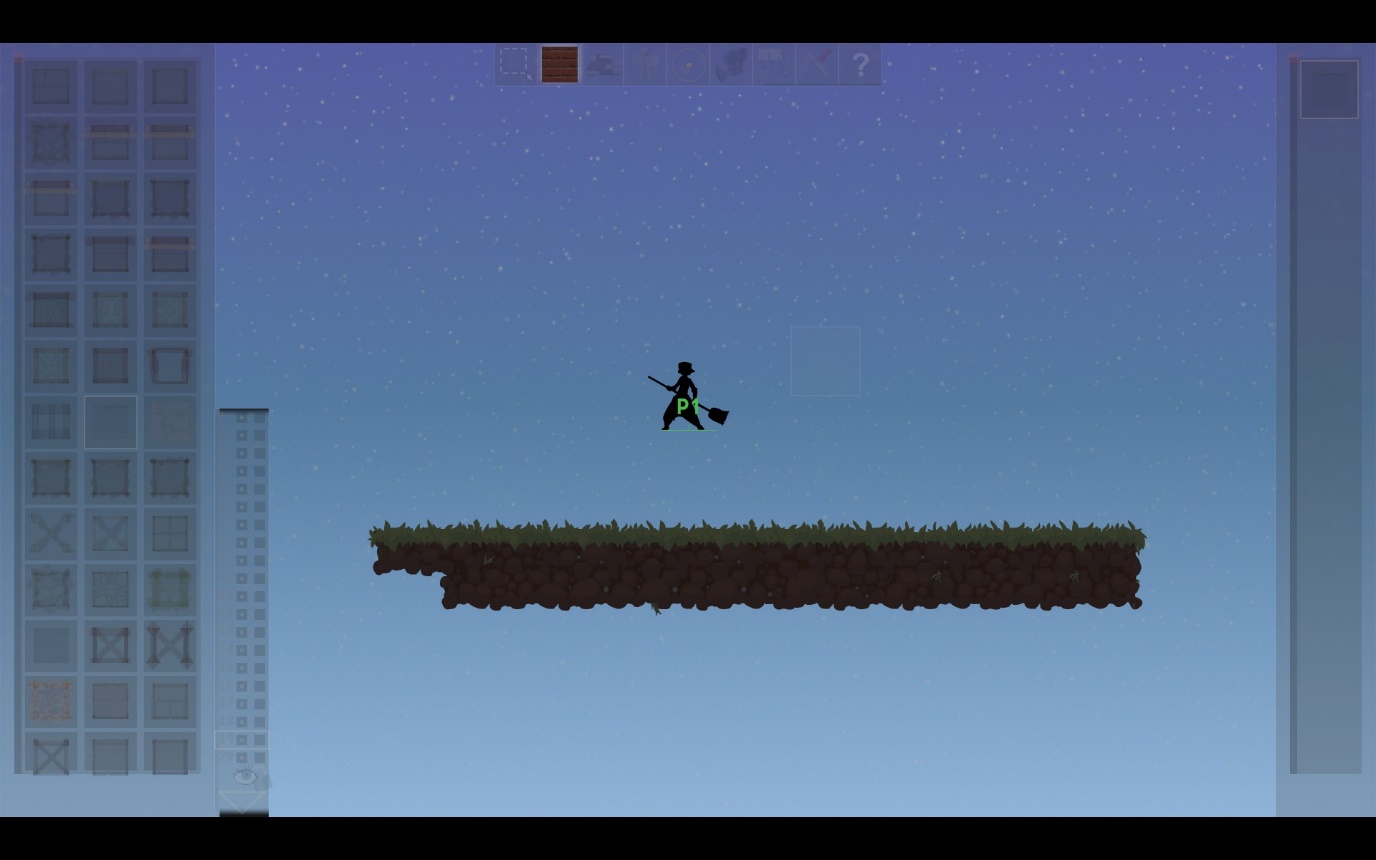


Figure 7

Figure 1 shows the main menu, which is a hub world. The player moves the character around to access the options available from the menu, such as playing levels and the level creator. This is a more interesting main menu than most games, but a lack of text and not all areas being on screen at once makes it difficult to navigate.

Figure 2 is the options menu accessed by pressing Escape in the hub world and is an example of the games user interface. This part of the options menu allows controls to be rebound for each of the 4 available player slots.

Figure 3 and Figure 4 show the rest of the options menus, detailing the available game configuration settings and video settings to the player. The audio and visual settings are rather limited, but the game options are useful for the player, allowing them to customize their experience a little.

Figure 5 shows a section of a typical level in the game. In the bottom left of the screen is a combo meter that allows the use of a special ability to destroy targets when full. Beside that is a timer for the current level, a useful indicator on how quickly the player is completing the level.

Figure 6 is the end of level screen. Here you are given a rank for how much of the level you cleared and one for how well you went about it. Your time is also displayed and compared to the global statistics for that level. You are then given the option to retry the level, watch a replay of your or someone else’s attempt, or continue to the next level.

Figure 7 demonstrates the level creator user interface. Levels are created on a grid-based system, with multiple layers to build up a parallax background, immediate background, the level scenery, and the player and enemies. Created levels can be played locally or shared online with the games community.

My experience playing Dustforce was mixed. The level design was superb, varying over the large quantity of levels, and was obviously built around the design of the game mechanics. The graphics served the game well, providing nice detail to the levels, whilst not distracting the player from the fast-paced gameplay. One aspect that is hard to show in this research is the games soundtrack. The music is gentle and calming, keeping your mind focused on the task at hand, but provides a nice ambient background. The audio cues also allow the user to engage with the game, with movement accompanied by relevant sounds.

Whilst the levels were well designed, I felt the default control scheme was counter-intuitive, leading to me struggling to complete levels. I would often fall into a pit because the game was picky about when I could use my double jump or dash, and this transpired into a frustrating spiral of failure. Once I remapped the controls from a keyboard to a controller I had a better experience, but that step should have been unnecessary. I also found the use of a hub world as a main menu to have more drawbacks than any benefits it provided. It was often difficult to find a new level to complete and added an unnecessary layer between the enjoyment of the levels. If there was any progression in level difficulty, it was hard to find because the hub world was very non-linear.

# System 2: N++

Development studio: Metanet Software

Release price: $20

Platforms: Initially PS4, expanded to Windows, Mac and Xbox One

Minimum system requirements:

* Core i3 / AMD A6 2.4 GHz processor
* 2 GB RAM
* 512Mb VRAM NVIDIA GeForce 8xxx series / ATI Radeon HD 3xxx series
* OpenGL 3.3 support
* 650 Mb available storage space

Development tools: Custom physics engine and vector graphics renderer

Estimated production cost: Unknown, but include PS4 and Xbox One license costs

N++, the second sequel in the N series, is the culmination of a decade’s worth of work by the two-person development team, Metanet Software. Around 7 years were spent on the development of N++, with a couple of years after release spent porting the game over to Windows, Mac and Xbox One. N++ built on ideas and designs used in the previous two games but sought to be superior. This lead to the use of a custom physics engine to achieve the desired character motion, and the creation of a custom vector graphics renderer, providing the game with a crisp, geometric appearance.

In N++, you play as a ninja working through episodes of 5 levels. To complete a level, you must get to the exit door, sometimes having to hit a switch to open it first, before time runs out. In between you and the exit are a whole host of enemies, such as mines, lasers and rockets, each able to end your run in one hit. Coins can be picked up in the level to extend the time available.

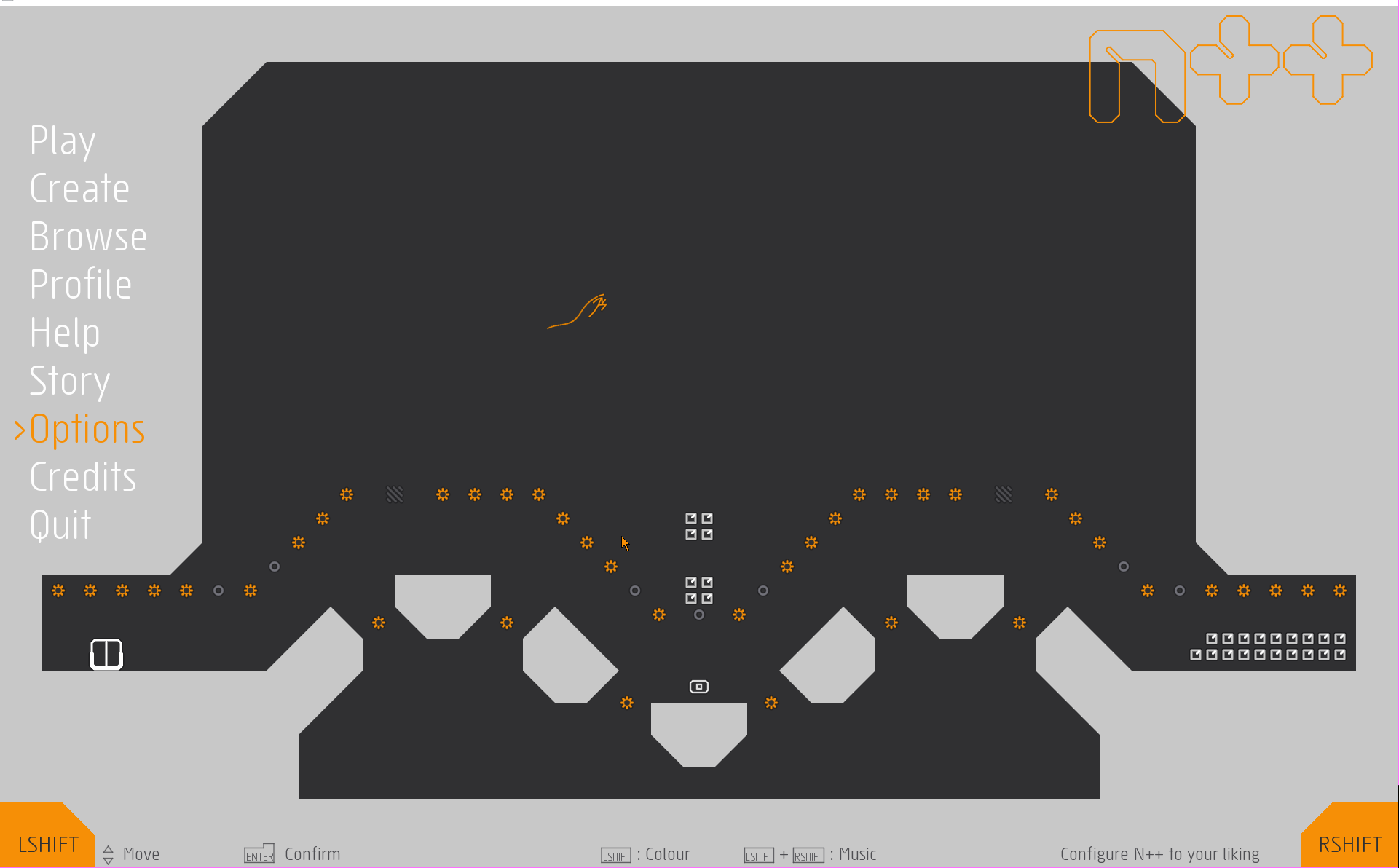


Figure 1



Figure 2

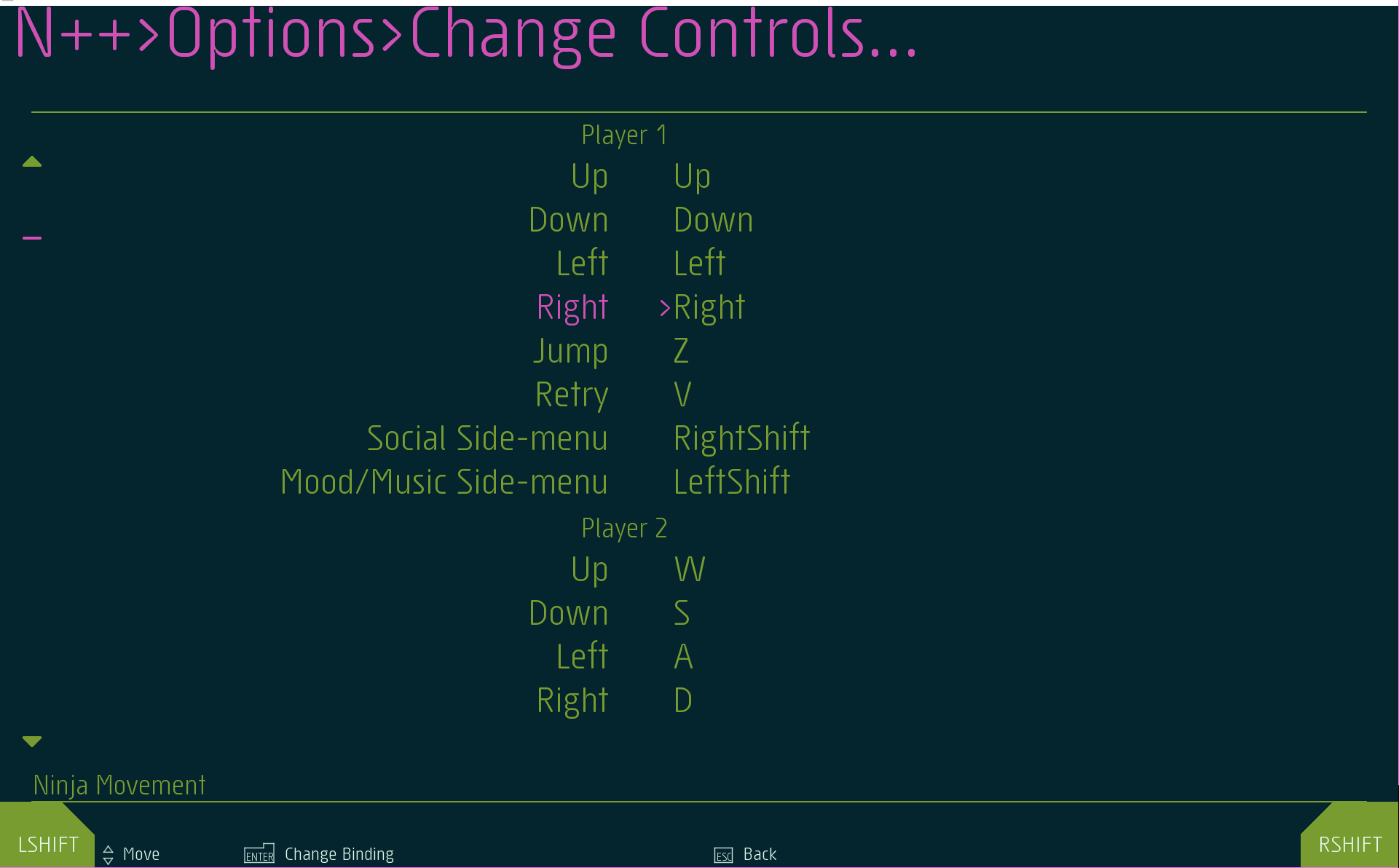


Figure 3

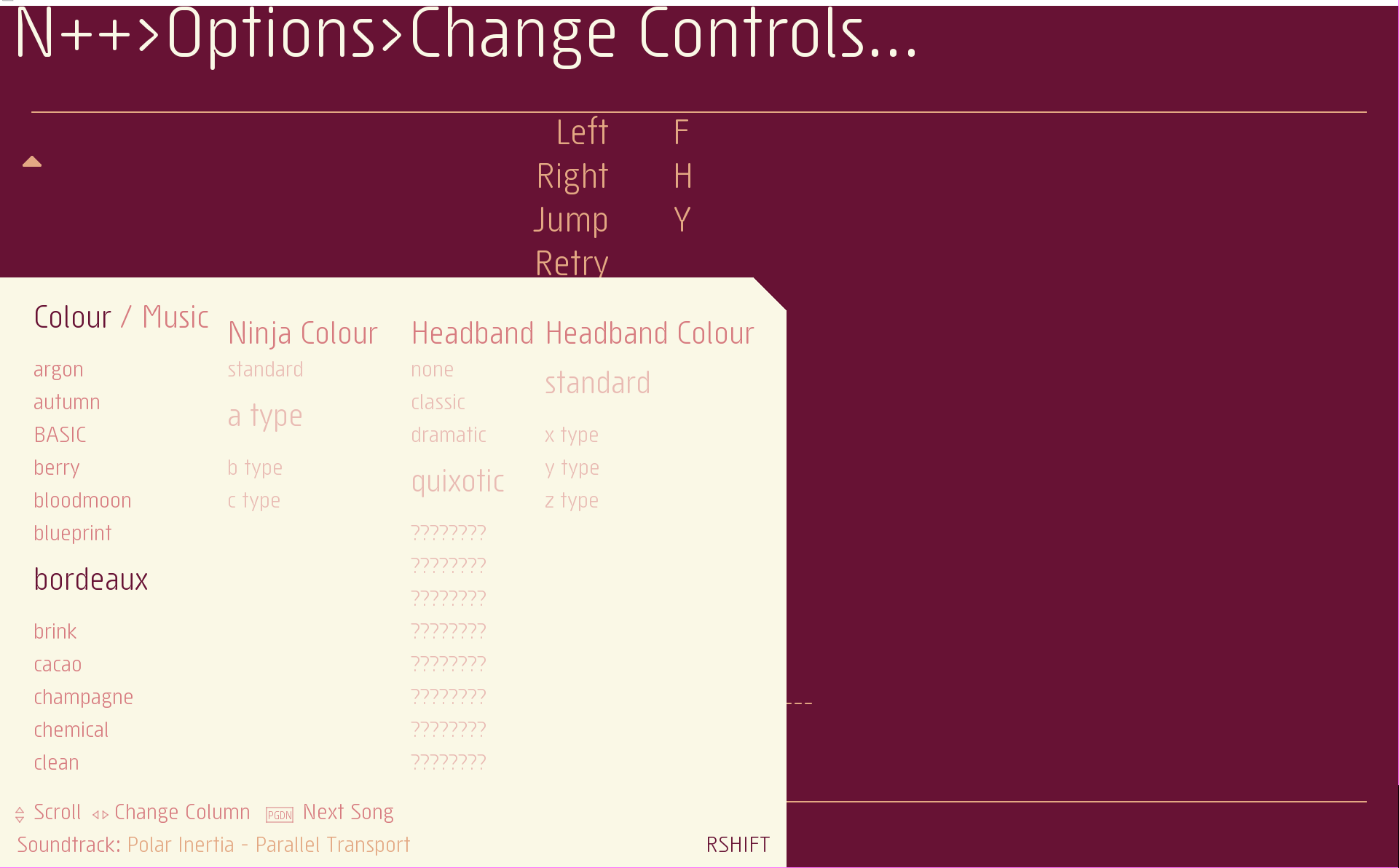


Figure 4

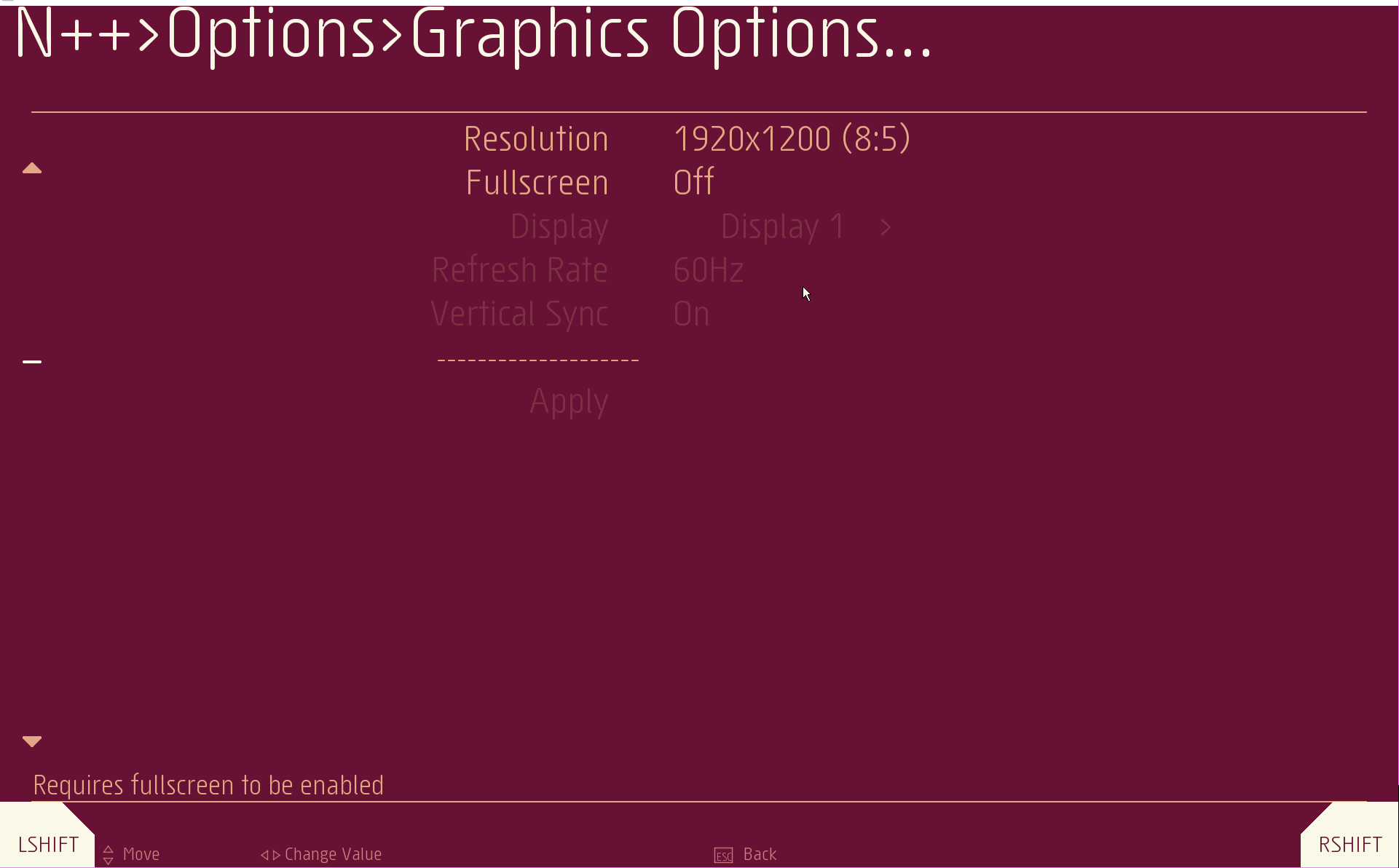


Figure 5

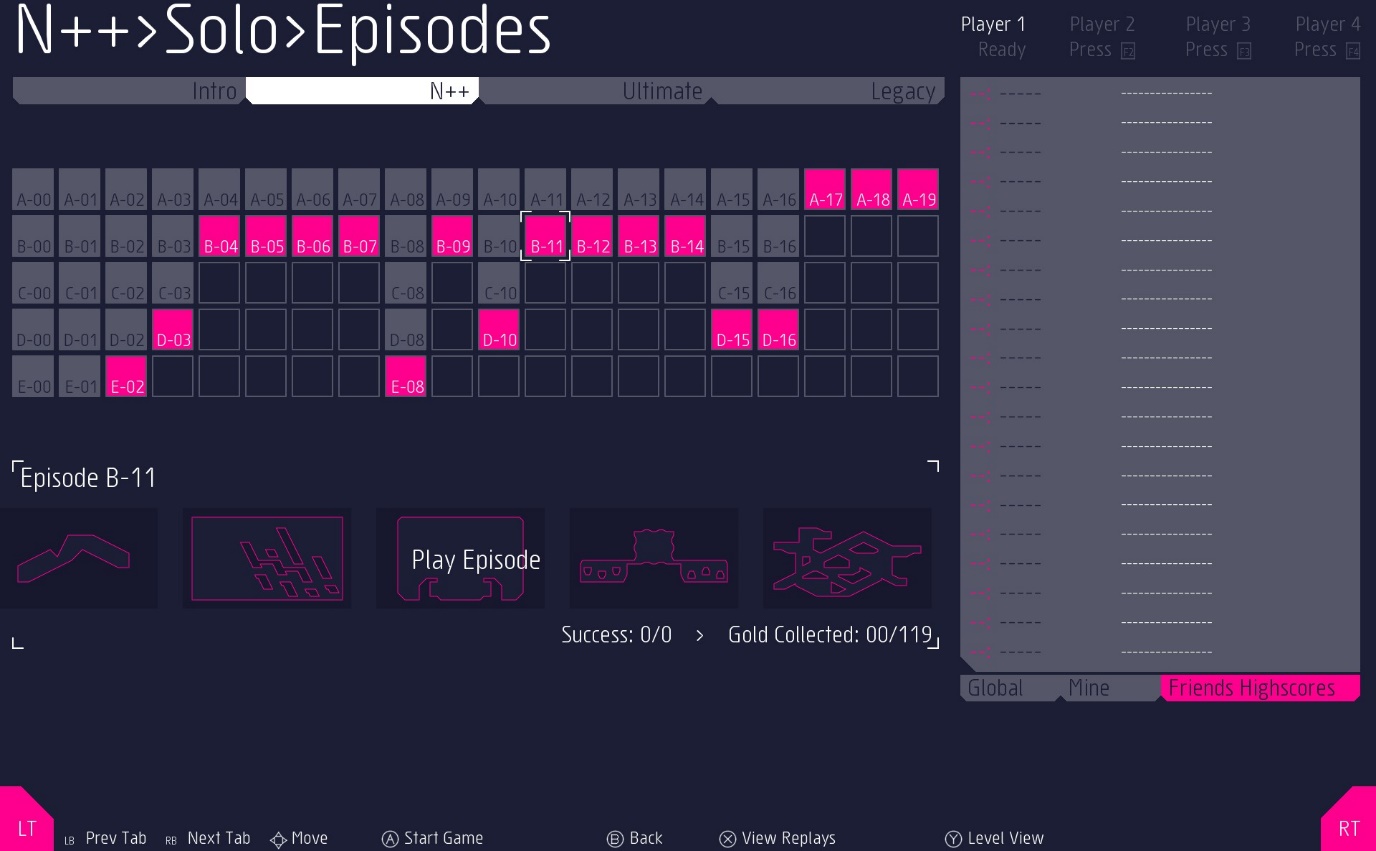


Figure 6

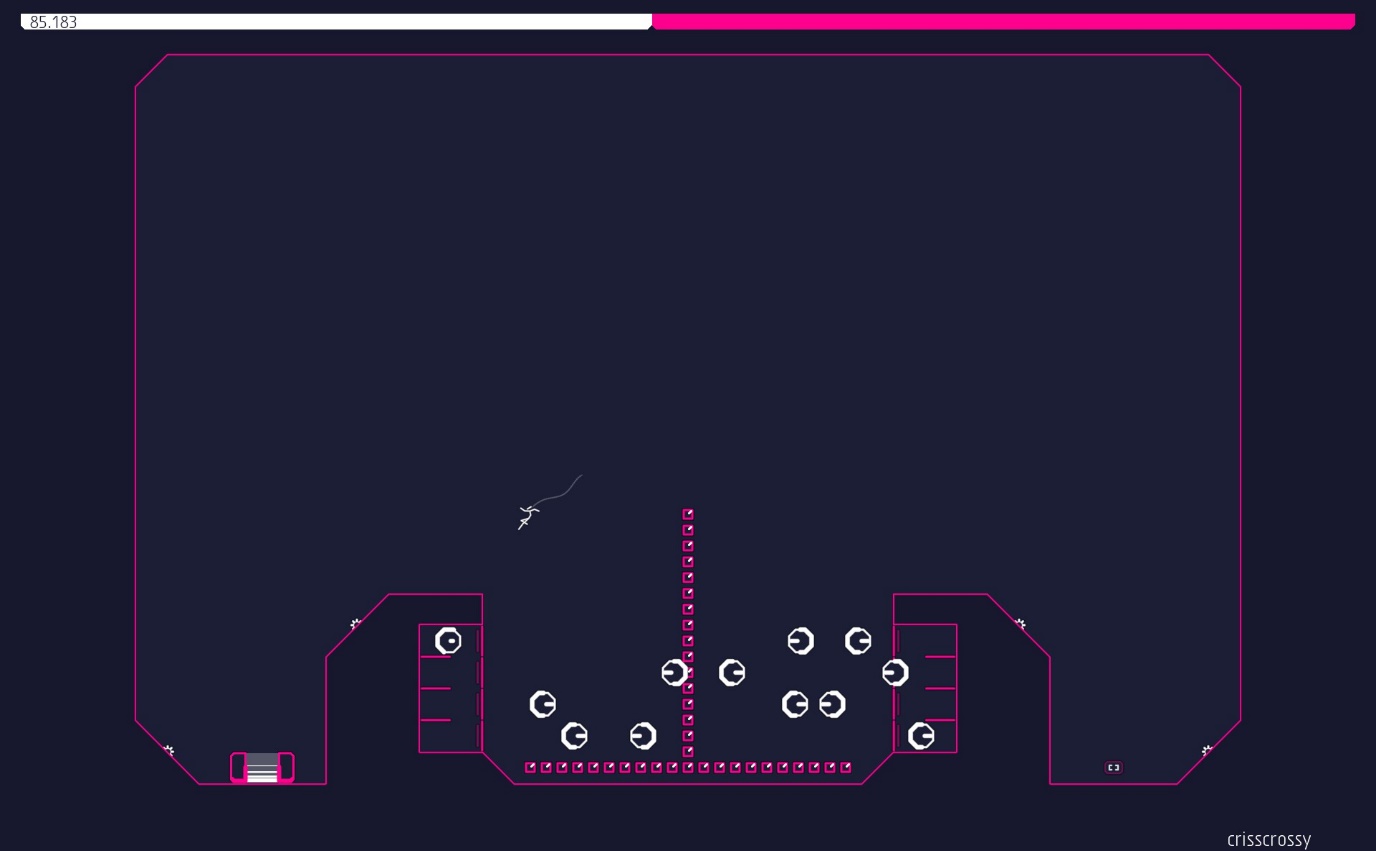


Figure 7

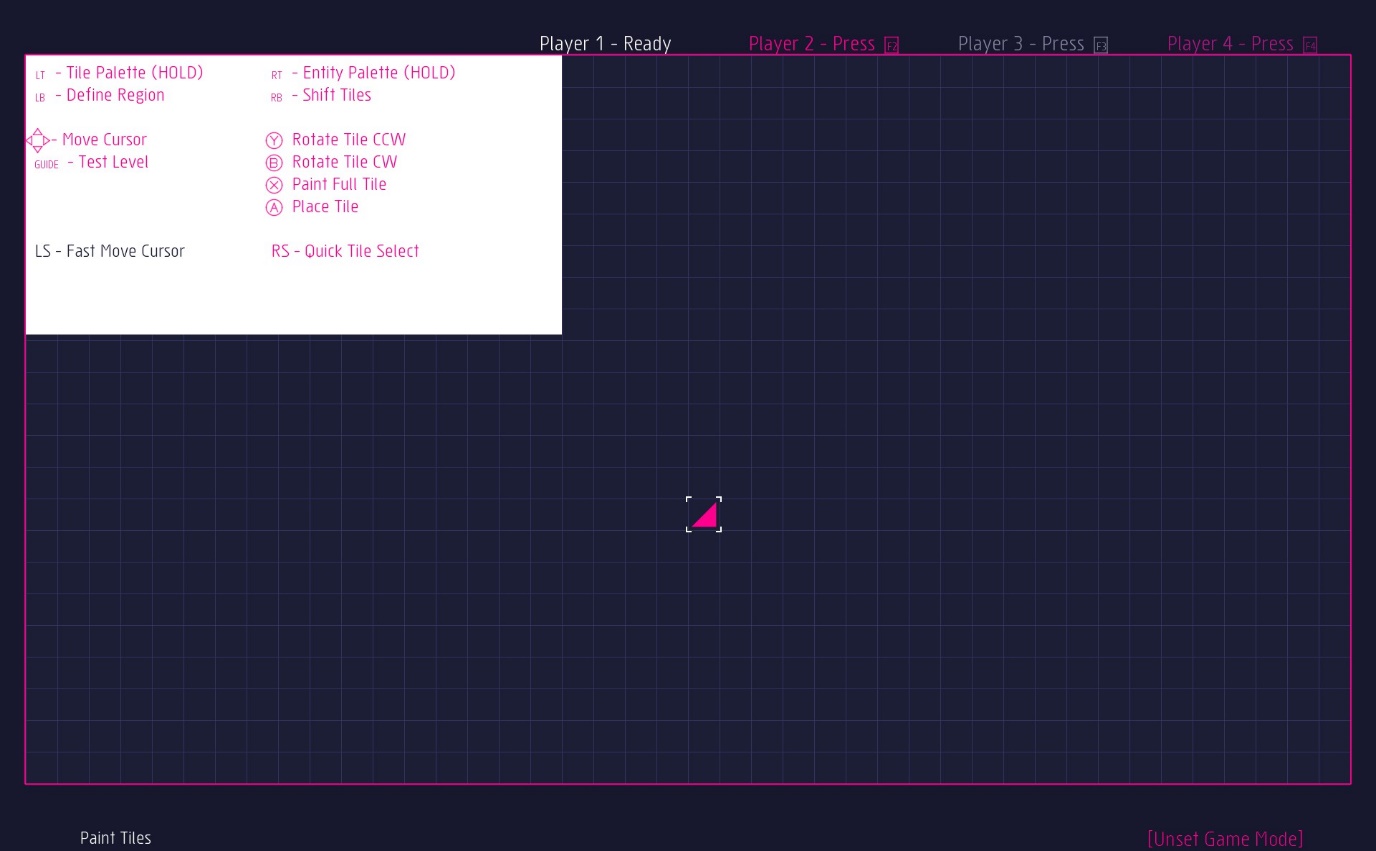


Figure 8

Figure 1 shows the main menu of the game, along with the style of the games UI. Button prompts are visible at the bottom of the screen, and the background cycles through replays of failed attempts at levels. The visual art style of the game is also prevalent. The menu choices can be cycled through top to bottom, and there is a sub menu when you press to quit, checking that you’re sure.

Figure 2 is a picture of the main options menu. Along with buttons to the typical options, a wide variety of more unique options are available. This picture also shows a different colour scheme for the game than the previous image.

Figure 3 is the control rebinding menu, allowing the user to customise the control scheme for each player. This menu changes depending on the most recently used input device. If that was a keyboard, it shows the controls for a keyboard user. If the device was a controller, it shows the relevant bindings for that controller.

Figure 4 shows part of the list of possible colour scheme choices, along with player character customisation menu. From this menu the user can access the control system for the in-game music.

Figure 5 details the graphical options available to the player. Whilst somewhat limited, the standard options are still there. An anti-aliasing option is not required due to the vector nature of the games graphics.

Figure 6 shows the level selection screen. The tabs at the top cycle the ‘type’ of level, whilst the central grid is the list of levels. Times are displayed on the right, either the quickest global times or the best times achieved by the user’s friends. Small outlines for the currently selected set of levels are displayed underneath the central grid.

Figure 7 shows a typical level, containing goals, enemies and scenery. The bar at the top of the screen is a timer that counts down over the set of levels, but which is refilled by collecting the pink square tokens that create the upside-down T in this level. Bottom left and bottom right are the exit door and door switch respectively. On either side of these two are white mines, that explode if the player touches them. The larger white objects in the centre of the image are an enemy type that follows a predefined path and kill the player if they come into contact with it.

Figure 8 shows the level editor with a blank level. A grid-based system is used to keep design consistency between levels, and the player gets to choose from a wide variety of tile pieces. The controls for the level editor are shown in a box that automatically moves to a different part of the screen when the player works in the current area.

You can tell N++ has 10 years of experience behind it. The game revolves around pure platforming, with levels designed to both interest and frustrate you. It’s a challenge, but it never feels unfair, because the controls are precise enough to put you in total control, and hence to blame for any mistakes. The clean visuals are eye catching, especially once combined with one of the many colour schemes available, but don’t get in the way of the platforming. The electronic music fits with the retro theme of the game, and the controls of the music player allow you to mute any songs you dislike on the track list. A large collection of levels and level editor provide plenty of content and ensure the game can be played for a long time.

One complaint could be that sometimes the menus don’t provide the optimum user experience, where something that seems intuitive, like scrolling to the bottom of a long menu when pressing up at the top, are nowhere to be seen, but these are minor issues.

# System 3: INK

Development studio: ZackBellGames

Release price: £3.99

Platforms: Windows, Mac and Linux

Minimum system requirements:

* 2 GHz processor
* 1 GB RAM
* 512 Mb VRAM graphics card
* Version 9.0c DirectX
* 30Mb available storage space

Development tools: GameMaker Studio

Estimated production cost: N/A outside of living expenses and GameMaker Studio license

The original release version of INK was made by one man over three weeks, totalling around 80 hours of work. This is a similar situation to myself, though I should be able to divert a larger amount of time to developing the solution. The game was made using GameMaker Studio, a proprietary game creation system primarily for developing games based on 2D graphics.

INK is 2D platformer where all of the terrain is invisible. When you bump into terrain, paint is spread on the surface and it becomes visible. When you double jump, ‘bullets’ of paint are released from the player so that you can uncover more of the world around you. The aim is to reach the end of the level, which normally involves opening the exit first.



Figure 1



Figure 2

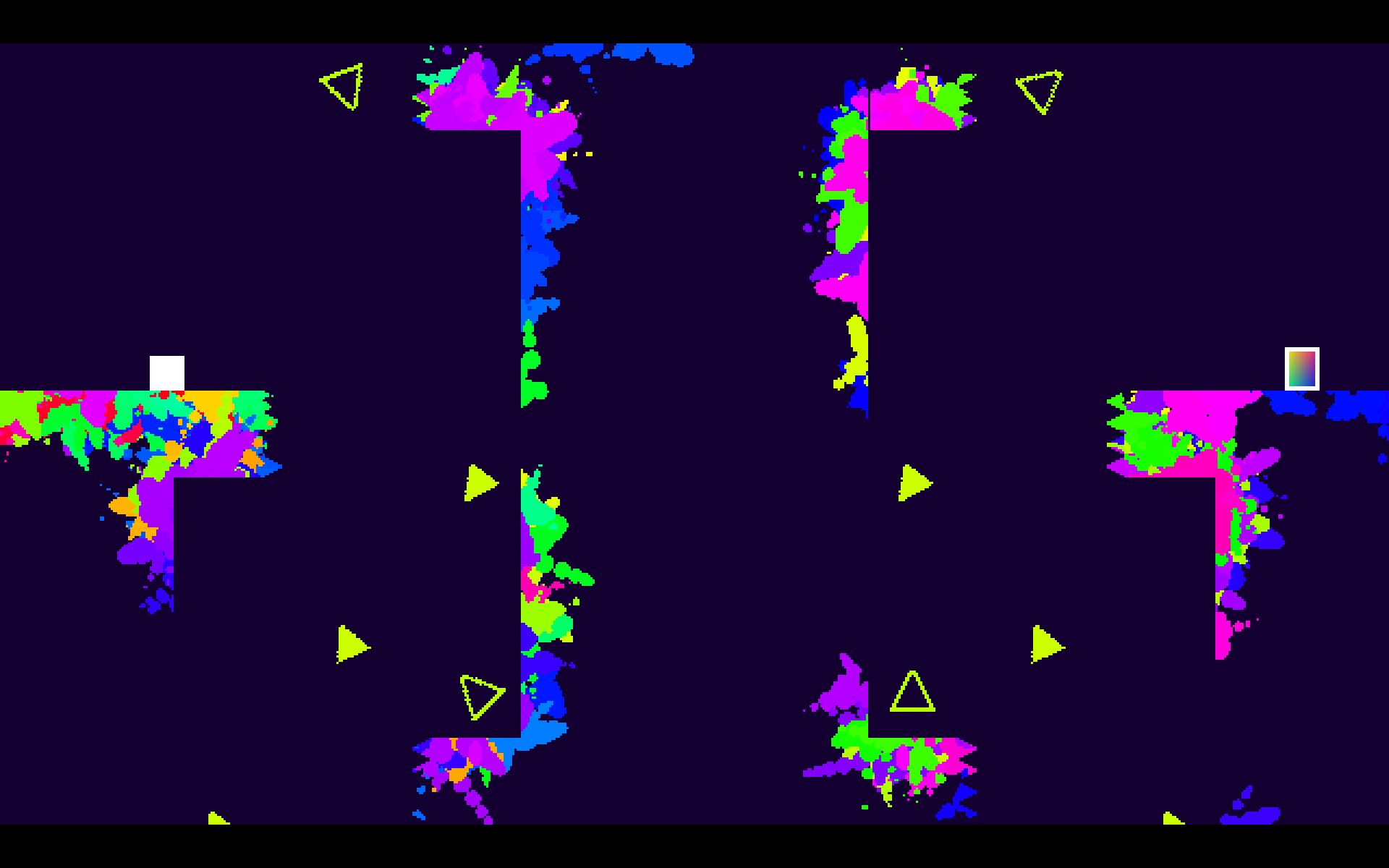


Figure 3

Figure 1 shows the menu visible once the game loads. This menu has limited options, with button prompts for both a controller and keyboard user shown together.

Figure 2 is the level selection menu. Here the user is able to scroll through the levels from left to right but can’t play levels past the one they are currently trying to complete. A small gameplay option of an in-level timer is available on this screen.

Figure 3 is an example of a level in the game. Splatters of paint cover the invisible scenery from the previous attempts of the player.

Inks core game mechanics drive the user to explore the levels and figure out what they are required to do to progress by themselves. From this, death is more of a learning exercise, as you find the path through. Once you know the path however, the one hit it takes for an enemy to kill you becomes frustrating, which at times is amplified by the seemingly imprecise controls.

At times the vivid art style was a too distracting, cluttering the screen and making it difficult to focus on immediate dangers. This was not helped by the fact that the enemies and projectiles blended in with the colourful surroundings, making them sometimes hard to see. The game has a total of 75 levels, which sounds like a lot, but offered only 2 hours of gameplay, making the game not very compelling to return to after completion. The level selection menu made it long winded to navigate from early levels to late levels and offered no indication of which level was which other than a level number. This lack of a level preview, or some other system, is possibly more a symptom of the “hidden level” mechanic than poor UI design.

# Research Summary

To summarise, my research shows me that this problem can and should be computerised. There are many similar products on the market, and from exploring a few of those, it is clear that a non-computerised system is not viable. All existing solutions were computerised and contained complex physics simulations that would not be possible without a computer. The current systems also demonstrated that the systems had to be robust and be of good quality, otherwise they will not appeal to their target audiences. Whilst testing those products, I experienced no crashes and the physics simulations ran smoothly, setting me clear targets for robustness and performance.

Due to the nature of my program, certain features and design choices are essential.

The program must centre around a physics simulation, which has certain requirements:

* The outputs of the system are determined by given inputs, and the simulation itself.
* It does not have to entirely reflect real world physics but should maintain a consistent representation of mechanical ideas.
* The user is able to input to the physics simulation to control a player character.
* Outputs of the simulation must be displayed graphically, with the graphical representation updated as regularly as the physics simulation itself.

My program will require a user interface with these features:

* The ability to navigate through the whole program in a logical way, with submenus separating different sections into categories.
* It must be intuitive to use with either a mouse, keyboard, or controller. It should have a graphical indication of which menu option is currently selected, and user-friendly features, such as navigating directly to the top menu option by continuing down from the bottom option.
* An options menu, with choices for graphical, audio, and gameplay settings, along with a changeable control scheme system.

In regard to storage:

* My solution will comprise of a collection of scripts, only some of which will be in use at one time. To minimise the impact on RAM, only the scripts that will need to be in use for that current part of the program will ever be loaded into memory.
* The final executable file should take up less than 1Gb storage space, as the largest existing system I researched only required 650Mb. This does not account for any files created and saved by the program.
* My solution will not require saving created files onto the hard drive unless I implement a level creator. In that case, files associated with the level creator will need to be saved to persistent storage.

For the hardware the solution should be able to run on:

* 2 GB of RAM as a minimum.
* A 2 GHz dual core processor or equivalent is a reasonable minimum for the solution to require.
* A dedicated graphics card.

It should be possible for me to build my solution to work on a variety of platforms. My user, Owen Barker, and most of the other people that will be able to provide feedback on my product run a windows operating system, and because a 32-bit program will work on both a 32-bit or 64-bit machine, the builds I release to my users for feedback will be built for 32-bit windows. No other software will be necessary to run the program.

Other features:

* The simulation must comprise of several game levels the user can play, able to be picked through a menu system.
* To account for different users of varying skills, an explanation of what to do or a tutorial should be available.

I will not be able to implement certain features, due to various constraints. In order to develop the game for platforms other than Windows, Mac, and Linux, I would require a license and development kit for that platform, both of which would be very costly. Implementing a leader board system, and a method for level sharing would both require a server system that I would run, or the use of platform specific networks, such as PlayStation Network or Steam. The former of the options is both expensive and impractical for me, and whilst the latter would be technically possible, the task is beyond my current skillset. Both options are beyond the scope of this solution.

# The proposed solution

**3.1.4 Specify the proposed solution**

**(a) Specify and justify the solution requirements including hardware and software configuration (if appropriate).**

Having identified and researched the problem, my solution has a set of requirements to fulfil.

|  |  |
| --- | --- |
| **User’s requirements** | **Explanation** |
| Game simulation | This is the main physics simulation that the bulk of the content of the game will be built into. |
| Levels | Different circumstances and initial setups of the game simulation, that offer the player an enjoyable challenge. |
| A goal, or target to achieve | Some way of completing the levels and progressing through the game. This could range from a place you have to get to in each level, objects you need to collect, or enemies you have to kill. Level progression could be controlled by having to complete the current level in order to play the next one, or a similar system. |
| Multiple system inputs | The ability to use different input devices to control the player character in the game simulation and navigate around the rest of the solution. The input devices I will focus on are a mouse, keyboard, and Xbox 360 controller. |
| Level creator | A place where the user can design, test, load and save their own levels. |
| User created levels | Made in the level creator, these would be locally stored on the user’s machine, and available to play through the game. |
| GUI menu system | The whole solution can be accessed by the user, using a graphical user interface, as opposed to the commend line. |
| Options menu | A submenu in which the user can change settings for the games audio, graphical, and gameplay settings. |

I have 2 possible input systems, keyboard and mouse, and a controller. Both of these systems need to allow the user to access the whole program without using another system, to maintain a good user experience. The process taken for different inputs will vary depending on the current state of the program.

Case 1: Navigating a menu or UI

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Processes** | **Outputs** |
| Interacting with a menu item | The corresponding menu or submenu is loaded. | The new menu is displayed. |
| Moving up or down the menu choices | The currently selected menu item is shifted by 1 up or down. | The next menu item is visibly selected. |
| Clicking the mouse over a menu item. | That item becomes the currently selected item. | That menu item is visibly selected. |
| Name for a created level | The name is added as a string to the level file. | The name is displayed in the player created level menu. |

Case 2: Playing the game

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Processes** | **Outputs** |
| Horizontal movement axis | The simulation accounts for the input and updates the characters velocity. | The character moves horizontally if possible. The sound effect of them moving is played during this movement. |
| Jump button | A check is made to see if the player can jump. If they can, the vertical component of the characters velocity is increased. | The character is seen to jump. A jumping sound and animation are played, and landing sound and animation played upon landing. |
| Pause button | A pause check is toggled, meaning the simulation starts playing if its currently paused, and pauses if its currently playing. | The game simulation is paused or resumed, and the pause menu is displayed or removed. Game music is replaced by pause music or vice versa as required. |
| Retry button | The level is reset back to its initial state, along with any items and music. | The reset level is displayed. |
| N/A | The simulation registers a collision that triggers the characters death. | A death sound is played, and the player has the option to retry or exit the level. |
| N/A | The simulation decides an action is performed by a Non-Player Character. | Appropriate sound and visual effects are played or displayed. |

All users of the system will have the same access, with the whole of the program available to anyone.

I will be developing the solution on both a school and home computer, using:

* Software
  + Unity 3D
  + Visual Studio 2017
  + Windows 10
  + Adobe Photoshop/Gimp
* Hardware
  + Computer (Lowest system specifications used)
    - CPU: Intel Core i5-6400 CPU @ 2.70GHz (4 cores)
    - RAM: 8.0 GB @ 2133 MHz
    - 64-bit OS
  + Mouse
  + Keyboard
  + Xbox 360 controller
  + VGA monitor (1920 x 1080)

The project has no budget, meaning I will not be able to have anybody else create game assets or sound effects unless they will do it for free. This limit, combined with a deadline of February half term, means I will likely only be able to create a prototype of my final product. This will contain most of the features to be present in the final release, though not all as refined as they could be. It will be expected that a user will have some experience with games, and so will figure out what to do, but the menu system should be intuitive, and a list of controls included just in case the user is not experienced.

**(b) Identify and justify measurable success criteria for the proposed solution.**

My success criteria for my proposed solution are as follows:

1. I am going to create a physics inspired simulation of a 2D world, that is displayed to the user graphically. For example, my player character will fall back down to the floor due to gravity.
2. All the processing for the simulation needs to be performed at the back end, so that the user can only see information they need to. This data hiding streamlines the users experience.
3. Create a system for the user to move the player character in the physics simulation, with the option to use either the keyboard or a controller. This will provide the user with options, allowing them to use the program in the way they find most comfortable and intuitive.
4. A selection of levels for the user to be able to play through. This will sustain my users interest in the game, allowing them to enjoy the game for longer.

1. Mechanics and non-player characters that provide challenge and make the game more interesting to play.
2. Create a user interface to navigate the various menus for changing in game options, selecting levels, and detailing information to the user during the game. This will be professional and consistent. I am doing this instead of a Command Line Interface, because a menu driven system is easier for most users to use, because that is what they are used to.

1. Create a level creation system accessible by the user at runtime, that will allow the user to create, edit and save levels that they can then play.
2. The program will be efficiently coded so that the final program takes up minimal space, and the processes optimised so that the program execution has as small an impact as possible on the host machine.
3. I will have a working solution by February half term.
4. I will have created a functional game that could sell if put on the market.