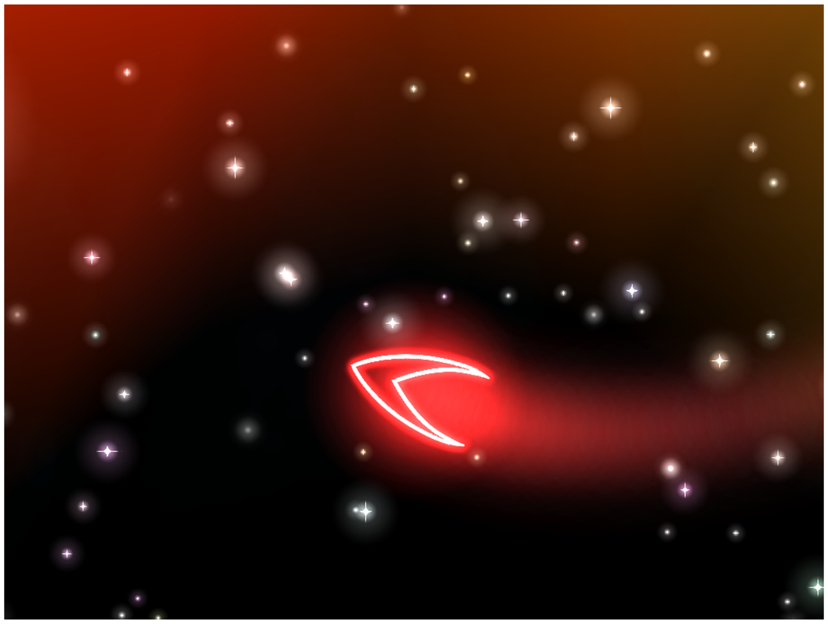
Compelling Interaction: An Investigation into Design for Play



Tomas Woodbridge

BACVA3 2008

Abstract

In this project I have explored some ideas relating to interface design and interaction. The study begins with some general thoughts concerning game design and videogames, and progresses into a discussion of interactivity and the concept of "ludic design". These ideas are then used to inform the design and development of a small game program, created in C++ using OpenGL and SDL. With this program I intended to implement the design principles discussed, and succeed in engaging users through use of interface and interactivity.

Contents

1 Introduction

Videogames are a complicated medium to study, as they encompass a broad range of disparate design disciplines. A typical videogame might involve art design, audio design, narrative, scriptwriting, gameplay and UI design, in addition to programming and software engineering. As someone with a keen interest in the medium, I have often wondered if there is some key element responsible for making a videogame a compelling experience. Discussing it with friends produces mixed opinions; some may cite the storyline and scenario, others the gameplay objectives and rules as the elements most fundamental to a game's appeal. However, certain games are likely to elicit a response along the lines of, "It's just fun to play".

This *'just fun to play'* is the aspect of videogames which I find most interesting. Of all the components mentioned above, it seems that this might be the only attribute of videogames which is fully unique to the medium. With this project I intend to investigate this area; what is this fun element, and can I recreate it in a game of my own? Is it distinct from the aforementioned components which make up a game, or is it a consequence of them all?

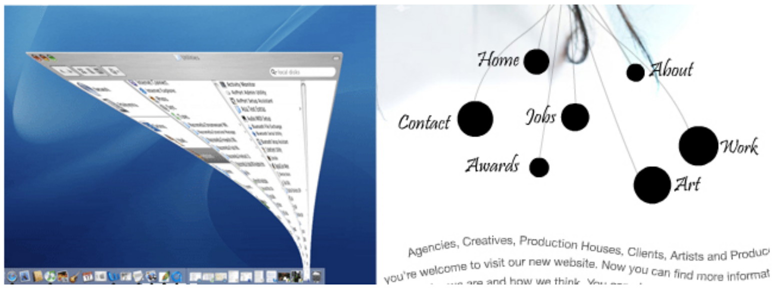
I suspected that this fundamentally "fun" element was to be found at the simplest level of interaction between the player and the game. For this reason, I began my research by looking into the areas of interface design, and human-computer interaction.

1.1 Preliminary Research

In his essay *Designing for Homo Ludens*[2], Bill Gaver defines "ludic design" as the idea of design for the purpose of play. This runs in opposition to the idea of design for utility; in the latter, the designer aims to overcome a practical problem, and consequently enable users to achieve a specific task. Speed, efficiency and ease of use are the common goals of utilitarian design. In contrast, ludic design champions ambiguity, surprise, and goalless interaction. Gaver argues that such type of design is not a worthless pursuit:

> *"We [humans] are characterised not just by our thinking or achievements, but by our playfulness: our curiosity, our love of diversion, our explorations, inventions and wonder. (...) Play is not just mindless entertainment, but an essential way of engaging with and learning about our world and ourselves - for adults as well as children."*

In the realm of computer software, this idea of ludic design is not isolated to videogames. As hardware has become more sophisticated and processing power has become an increasingly disposable resource, these ideas have infiltrated many areas of software and interface design. Apple is an example of a company who has leveraged this type of design to great success. Since 2000 their operating systems have incorporated many interactive elements which are subtly playful in nature: windows appear to slurp in and out of the taskbar; icons balloon in response to an approaching cursor; the entire screen is pulled smoothly into darkness upon entry of a media player. These cosmetic changes all work to imply that the screen is a physical 3D space, strengthening the metaphor with real-world interaction and making the OS feel more intuitive. However, these cosmetic additions also mean that the computer is no longer engaging users on a purely functional level; it has introduced elements of surprise, exploration and ludic appeal.



Agencies, Creatives, Production Houses, Clients, Artists and Produc
you're welcome to visit our new website. Now you can find more informat

Figure 1: Ludic design ideas in evidence in operating systems and web design.

Modern website design has also been informed by these ideas. A simple example can be seen at the website of animation company Bitt Animation[1]; rather than remaining static, the buttons on this page appear to respond to the mouse in a tactile manner, providing a brief thrill to visitors exploring the site.

I identified this type of design as the source of the fundamentally "fun" element which seems to exist in some videogames. The question which followed is: can this element be isolated? Would it be possible to strip out all other aspects of a videogame, such as gameplay objectives, context and storyline, and create a compelling experience through use of ludic design ideas alone?

2 Aim

The aim is to make a simple program which uses ludic design principles to engage the user. The software must be designed so as to invite curiosity and playful interaction from users; if successful, the program should compel users to uncover all of the mechanisms of the game, without resorting to other avenues of involvement such as narrative, objectives or stated instructions. Methods for prompting this behaviour from users will form the main focus of research.

In the process of creating the game, I also intend to develop my own understanding of game design and software development.

3 Design

3.1 Interface

My research lead me to believe that tactile feedback is the key to engaging users on the interaction level. As such, the idea of 'tactility' became the guiding principle of my design.

I chose a side-on perspective, allowing for a 2D representation of objects which could respond predictably to gravity and collisions. Physically-based behaviour seems to be appealing in an interactive context; it implies a tangible space, which users can relate to instantly. In addition to this, it lends a consistency and uniformity to object movement which users are able to predict and anticipate.

Figure 2: Despite its semi-abstract visual style, Sony Computer Entertainment's *Locoroco* manages to create an instantly convincing and tangible gameworld through its use of realistic physics.

The goal of my interface design was that users should be able to master it intuitively without instructions. As such, it was necessary that no part of the control scheme was mapped to the keyboard; any function hidden on a specific key would necessitate the need for a manual. Players needed to be able to control the game intuitively via the mouse alone. Consequently, my goal was to design a single-button control scheme which would depend only on the movement of the mouse and the left-click button.

I drafted several ideas for control schemes which could achieve this.

My first idea on this issue was very straightforward. I envisaged a straightforward click-and-drag scheme, in which the user can move the player entity by pushing the mouse toward the edge of the screen. The further the mouse would move from the center of the screen, the greater the acceleration of the player entity. Other actions could be achieved by swiping the mouse across the player entity in a specific direction; for example, a jump could be performed with an upward swipe.

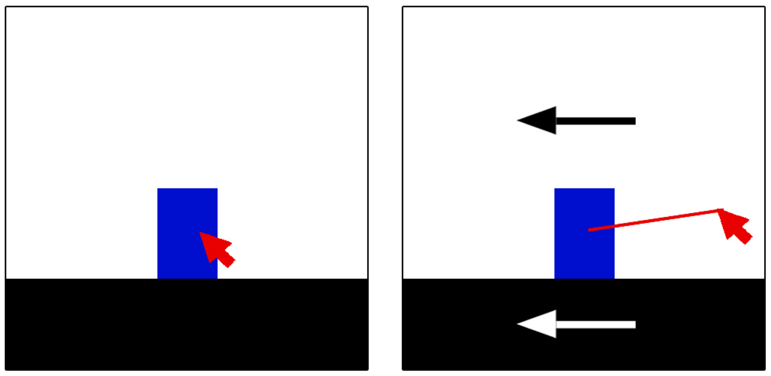


Figure 3: First idea for a mouse-driven control scheme: ‘walking’.

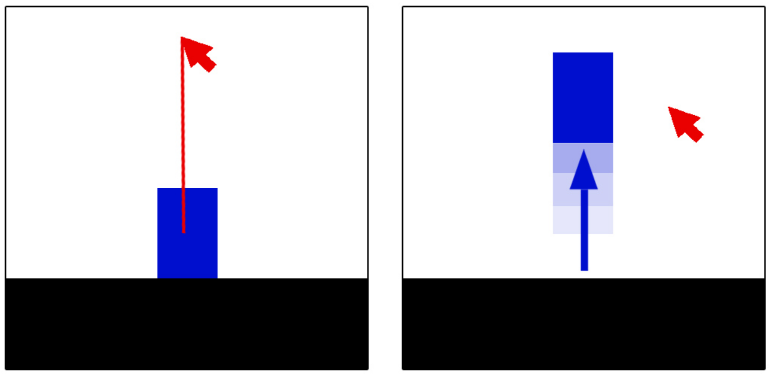


Figure 4: First idea for a mouse-driven control scheme: ‘jumping’.

I was aware that this control scheme might not be so successful in practice, so I noted several variations on the scheme which would need to be tested during development. One of the more distinct ideas was a rotational control system; players would be able to build up velocity either left or right, by rotating the mouse around the center of the screen either clockwise or anticlockwise. This scheme was likely to feel fairly tactile; the winding motion of the mouse would resemble the real-world activity of winding up a spring or an engine. In addition, the gradual acceleration of the player object would create the illusion that energy is being transfered from the player’s hand to the on-screen object.

These ideas could not be approved until the project reached a stage where they could be tested.

3.2 Visuals

Attractive imagery is always helpful when engaging an audience. However, creating statically “pretty” images was not my aim with this project. A good-

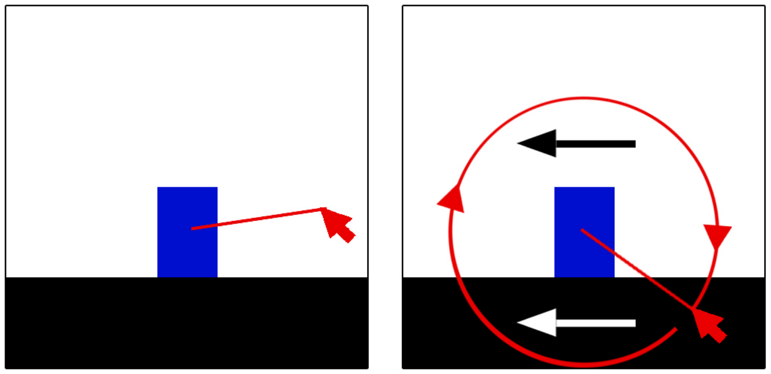


Figure 5: Developed control idea: rotational mouse movements build velocity on the player object.

looking piece of software would not necessarily be more successful at engaging users on an interactive level than a crudely-drawn one: in my own experience, I have seen players quickly lose interest in attractive retail videogames, quitting even before the interactive components of the game have been introduced. These games were relying on visual and narrative appeal alone to sustain interest during their opening stages; and in doing so, were arguably failing to leverage the unique potential of the medium in order to engage users through interactivity.

Due to this idea and constraints of time and ability, I chose not to aim for showstopping visual appeal. Instead, I decided to approach the visual design with the aim of emphasizing the ludic appeal of the game. It was my belief that graphical effects have the most impact on the user when they are tied closely to his or her actions; and as a result, I decided that every on-screen event should occur in response to user input. This would be applied consistently, such that every visual element should be tailored to highlight the interactions between the user and the program. I would try to avoid including any static screen elements which would not respond in some way to user input.

Following on from this, the idea occurred to me that every element visible on the screen should be present as a direct consequence of the player's actions. This would create the strongest connection between the user and the on-screen imagery. It followed that the game's opening screen should include the bare minimum number of elements; starting as a blank canvas, the game space would populate with imagery as a result of the user's continued interaction.

This idea also lead to solution of another problem. Typically, a retail videogame will bombard users with new information: a HUD, on-screen prompts and/or a full tutorial are all expected within the first 15 minutes of the game. For many users, particularly those who are unfamiliar with the grammar of videogames, this process might seem to run counter to the concept of 'play'; rather than play, this appears to be heavily structured *work*. Dedicated audiences will not be phased by this; but those who are less involved might find it

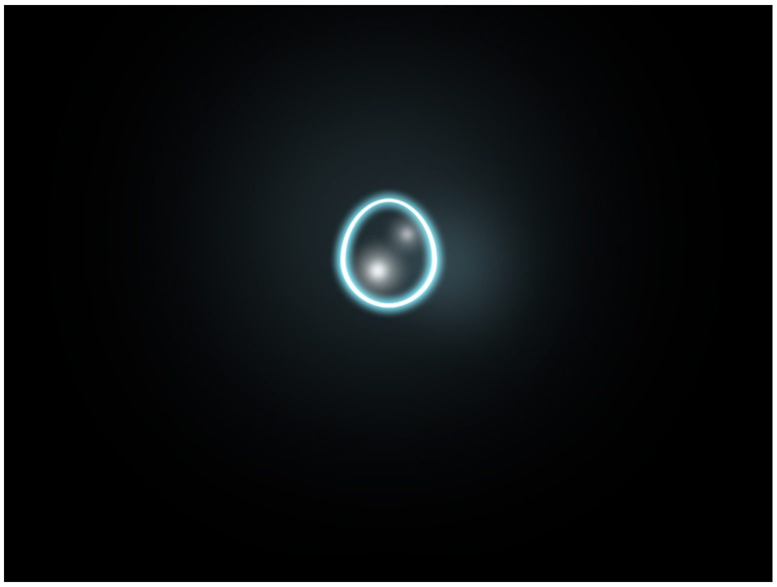


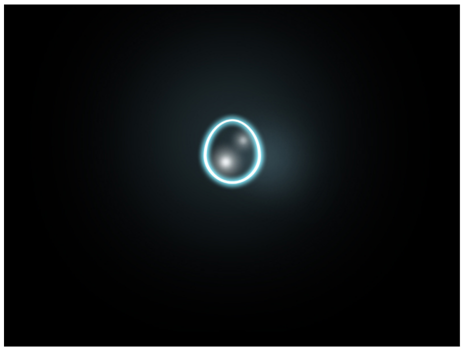
Figure 6: Concept art for the design of the game's opening screen.

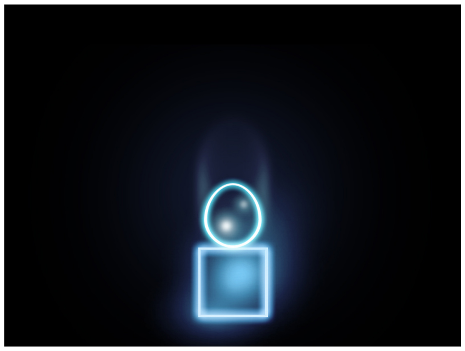
off-putting, to the point that they give up on the game altogether. By introducing screen elements one at a time, I hoped to avoid the information overload which could otherwise turns users away.

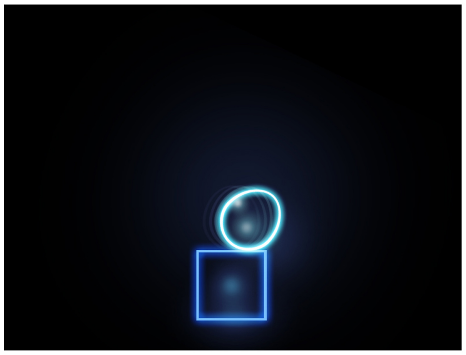
When deciding how entities within the game should appear, there were several problems to consider. If I was to apply a recognizable context or theme to the game, the element of user-lead interaction would have been partially lost: familiar imagery would immediately bring with it associations of real-world interaction, narrowing the range of approaches taken by users in their attempts to operate the game. In order to avoid 'context clues' influencing user attempts at interaction, I decided that all visual elements in the game should be abstract and geometric. This style would also allow me to test the ludic design principle of *'projection'* - the idea that ambiguity can be intentionally woven into an artifact's design, with the purpose of eliciting "richer responses and inspiration from users"[3].

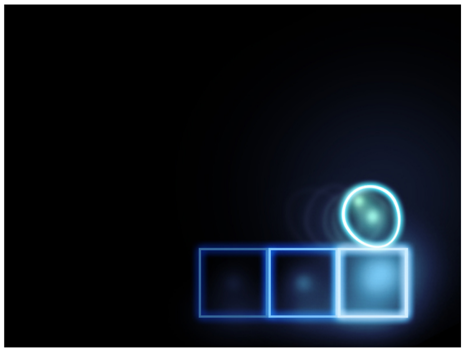
3.3 Final Design

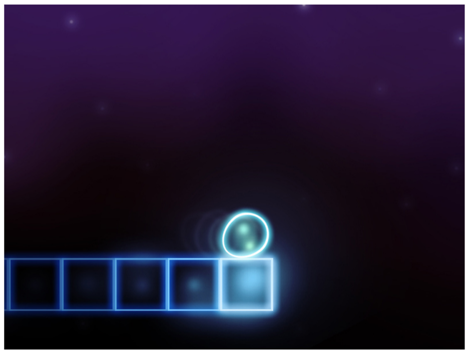
The final design is a 2D platformer-style game with an abstract graphical style. Emphasis is placed on creating immediate and tactile responses to player actions; every element on-screen is to respond visually and/or aurally to a mouse click. These responses may be small or inconsequential, but they should intrigue users and encourage further experimentation. The interface consists of the single-button control scheme described in the preceding section. The graphical style is abstract, with in-game entities represented by geometric shapes.

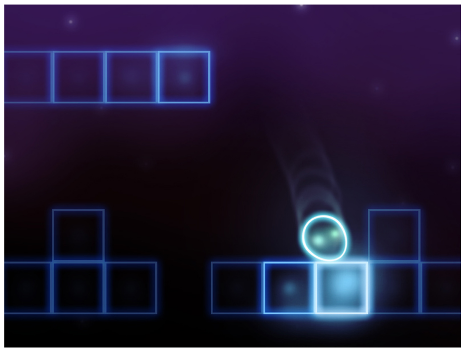

Opening screen: no clues are apparent to the player.

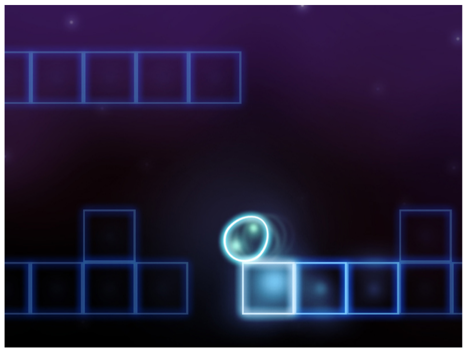

Clicking on the shape causes it to drop, revealing a second shape on the screen. This gives a small amount of new information to the player: it implies that we are viewing two objects from a side-on perspective, in a space where a gravity-like force is active.

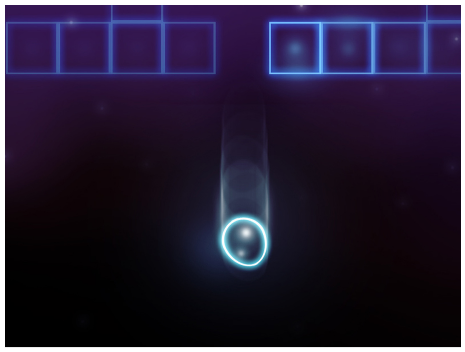

Intrigued by the effect of the first click, the player will click again in an attempt to prompt more action. This time, movement will be harder to provoke. The unstated challenge here is to work out how to manipulate the mouse to cause more activity on the screen.

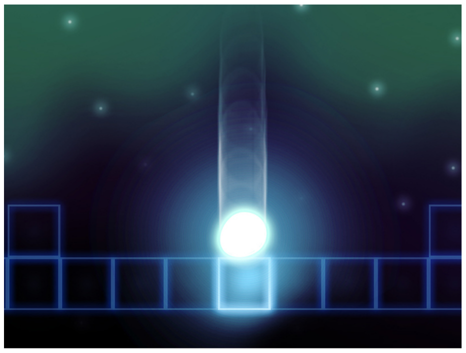

If successful, the object will move across the screen. The player is rewarded with immediate feedback: a new light appears on the screen with each step of progress. Each visible shape will response visually and/or aurally to a mouse click.

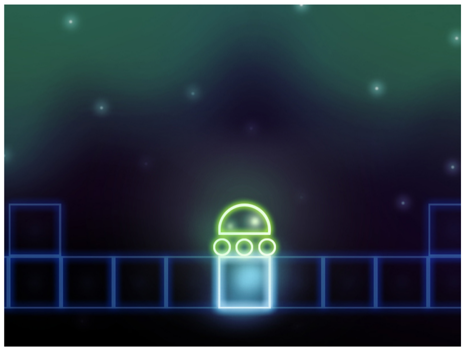

Now the camera moves to follow the player object, revealing a parallax backdrop which suggests depth and dimension. At this point a scene has been established, in which every element has appeared in response to the player's input.

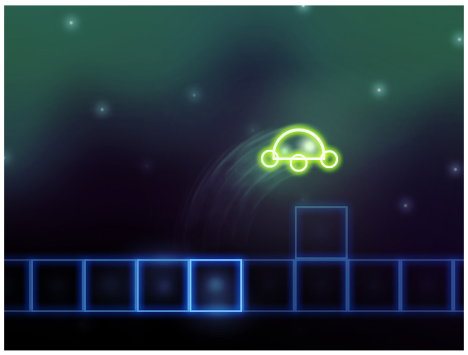

The player object runs into an obstacle. This reveals new information about the space depicted, and also levels an implied challenge to the player to negotiate the obstacle and continue.

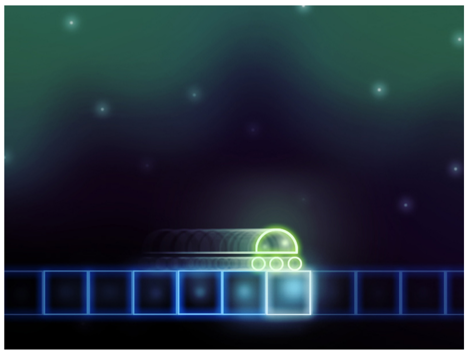

The player must successfully manipulate the controls to reverse the object's velocity

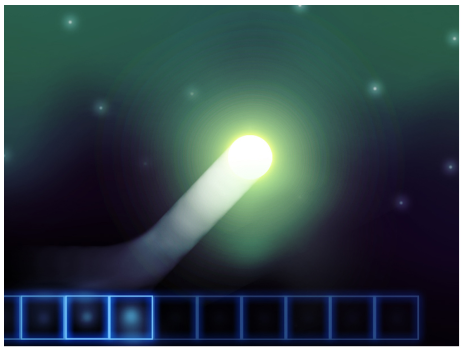


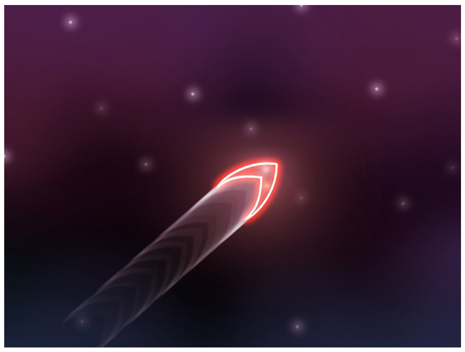



The result of the player's exploration is the appearance of a new object. This hints at new control possibilities for the player. The presence of impassible obstacles on either side of the screen present an implied challenge.


Clearing the obstacles is the first real text of dexterity; the player must build up horizontal velocity, and switch this momentum into vertical velocity at the correct moment.


The player is now free to move on an infinite plain. Unlimited space means they are able to use the control scheme mastered previously to achieve high speeds. Visual indicators such as motion trails and sparks will reward this behaviour.


A player who builds up sufficient speed will be rewarded with a second transformation.


The final form of the player object introduces a new control scheme, which must be quickly interpreted.
The player has now been lead from a blank screen through three distinct control schemes, without any stated instructions or objectives.

4 Implementation

The game was created in C++ using OpenGL and SDL. A fairly simple object-oriented structure was used, although the development of this structure took some time due to my relative unfamiliarity with object-oriented programming practices. The final program structure incorporated three main components. Firstly, a 'Scene' class was used to contain and manage all of the non-graphical data about the game's current state. Every game entity with a position or screen presence was to be created and destroyed within the Scene object. With each iteration of the game loop, the main function would call the Scene object's 'Update' member function; this in turn would update every entity in the scene.

A 'Draw Engine' class was created on the same level; this allowed all of the OpenGL function calls to be contained within a single object. The Draw Engine object would access the contents of the Scene object by reference, and use the data stored there to build a 3D environment and draw it to the screen. Finally, an 'Input Handler' class was created, containing all of the SDL calls needed to interpret user input. Like the Draw Engine, the Input Handler object would access the Scene object by reference, and directly edit the contents of the scene in response to player activity.

4.1 Collision Detection

Due to the limited time available to construct a working engine, I decided to limit collision detection to bounding-box collisions.

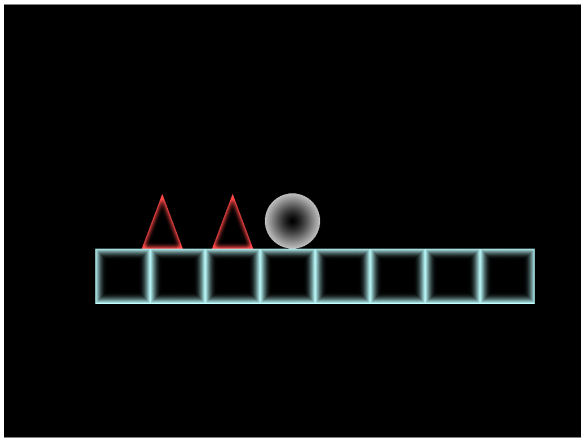


Figure 7: Early bounding-box collision tests with basic physics applied.

I developed a basic hierarchical system for handling scene collisions. With each iteration of the gameloop, the main function would call Scene::updateScene(). This function would iterate through every dynamic entity in the scene, and pass each entity by reference to the template function Scene::performCollisions(). This function would perform all relevant collision checks on the entity. If an

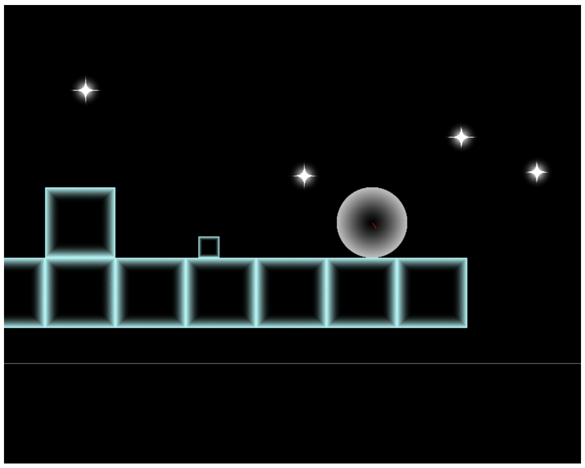


Figure 8: Early test: entities of non-uniform scale, behaving with class-specific responses to collisions.

intersection between two entities is detected, the member function 'collisionResponse()' is called for both entities, with the intersection vector passed to the entities as an argument. The entities would then be responsible for altering their position, velocity or colour in response to the collision.

The advantage of this system was that each derivative of the parent 'Entity' class could respond uniquely to collision events detected within Scene::performCollisions(); simply overwriting the 'collideResponse()' function of the class would provide a unique collision behaviour for that type of entity. This meant that boxes could be made difficult to push, falling heavily and landing without any bounce; whereas sparks could spring energetically away from collisions, with added random values influencing their velocity.

The collision detection I implemented was functional, but very poorly optimized. On every frame, each dynamic collision entity would be checked for collisions against every other dynamic collision entity; this meant that the number of checks performed would increase exponentially as more objects were added to the scene. In an attempt to optimize this, I implemented some simple broad-phase collision detection. This was a basic tile grid system[8], in which the address of each static tile entity was added to a 2D array during the construction of the scene. The array element selected for each tile was determined by the x,y coordinates of the block in scene-space. This allowed any entity to dereference the grid array using its own scenespace coordinates; thus accessing whichever tile entities exists in the nearby area. The result is that only four collision queries were needed when colliding a dynamic object against an entire scene of static tiles.

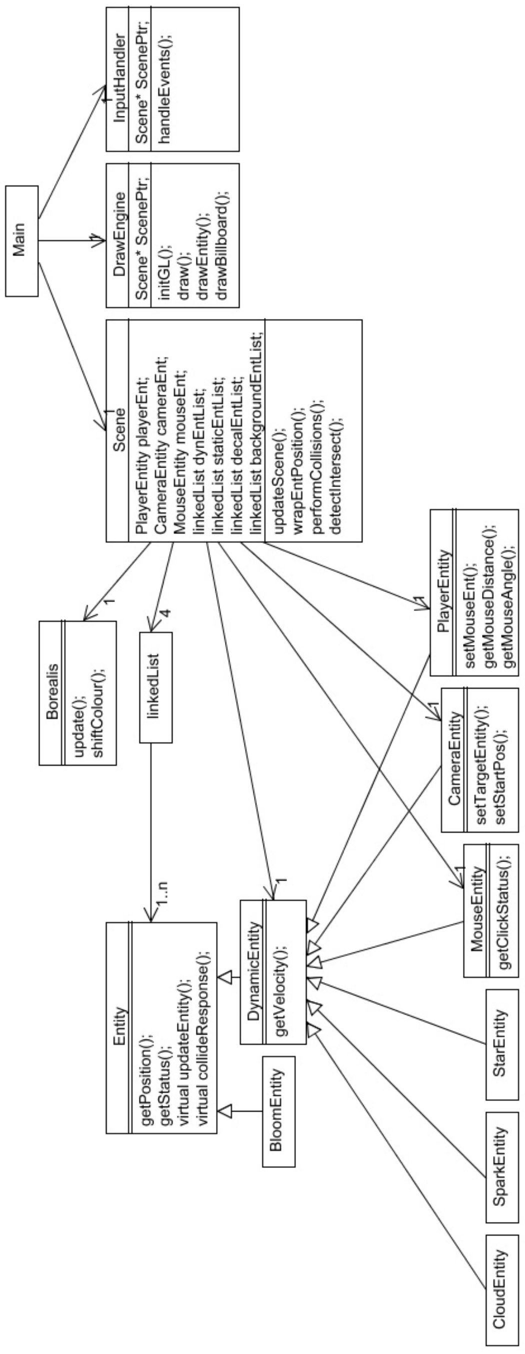


Figure 9: Simplified UML diagram of basic program structure

4.2 Stage Design

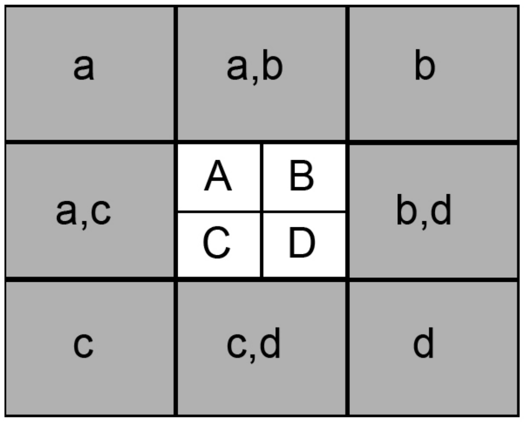


Figure 10: The system used to create the impression of an infinite play space. Depending on which A, B, C, D quadrant is currently occupied by the player, the playing area is drawn three additional times along the edges of the stage.

One of the important aspects of the design was that the player should be focusing on their immediate interactions with the game. I did not want to present the player with any potentially tedious routefinding or navigational tasks. With this in mind, the game's space had to be designed in such a way that there was no 'wrong' way to go; players should never be lost, trapped or forced to look for an exit. In effect, there should be no dead-ends.

In order to achieve this, a system had to be developed whereby the small, finite game space could be seamlessly 'wrapped' from left to right and top to bottom, creating the illusion of infinite space. This would permit players to run or fly infinitely in any direction, while keeping the amount of data involved compact and manageable.

I implemented this in two stages. The first stage simply required entities which passed beyond the boundaries of the stage to be moved to the opposite side of the stage, while keeping their velocity intact. This was straightforward to implement, but of course appeared disorienting when the player object crossed the boundary. The second stage was to change the behaviour of the Draw Engine class, such that it would draw the entire contents of the scene more than once each frame; first, it would draw the scene in the correct worldspace position; next, it would draw the scene again in a full transpose to the side of the original draw. This created the illusion that there was no scene boundary; when approach the edge of the scene, the other side of the scene would come into view as though it was a single continuous space.

The effect of this was successful when moving objects across the scene boundary. However, it introduced a series of problems relating to collision detection.

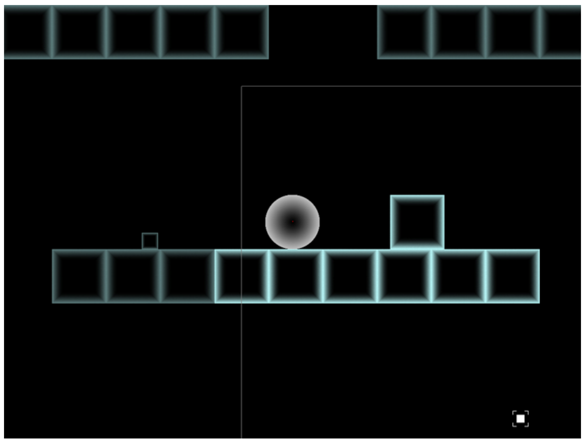


Figure 11: The player object approaches the boundary of the stage. Here the stage boundary is drawn as a white line, and transposed sprites are dimmed.

Entities at opposite ends of the stage would appear to be positioned alongside each other; but when they appeared to touch, they passed through each other without detecting collisions. Obviously this was because the entities were not actually alongside each other, but at opposite ends of the stage. This required an upgrade of my collision detection function; in order to correctly detect collisions, it was necessary to temporarily transposes entities away from the scene boundary during the detection phase.

4.3 Graphics

I intended the keep the game's visuals as close as possible to the style developed in the planning stage. This would require the implementation of glow effects and trails.

I investigated some procedural methods for implementing a full-screen 'bloom' filter. However, it became apparent that it would be more economical both in terms of game performance and project time to emulate the bloom effect using sprites. I decided to implement 'bloom' as an attribute local to each screen entity; when an entity is drawn to the screen, its bloom sprite is drawn on top of its primary sprite, create an aura-like effect. This was a beneficial approach, as it allowed the glowing appearance of each screen element to be tweaked individually on a per-object basis. The colour and brightness of an entity's bloom sprite would be determined by colour values saved within the entity; this allowed the glow of an entity to be changed in direct response to a collision or a mouse click.

In practice, I found that the effect was unsatisfactory. Transparent sprites layered on top of each other did not appear to give the impression of a 'glow' or bloom. However, I found that by changing the OpenGL blending mode I could

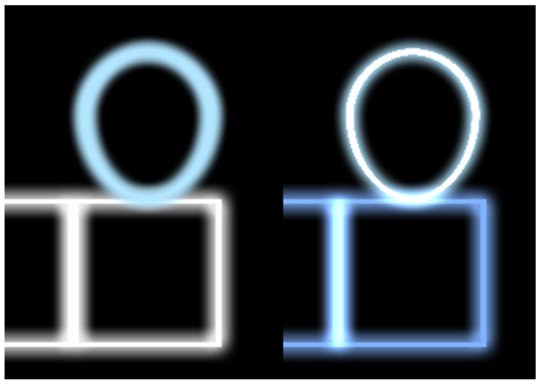


Figure 12: Left: two sprites combined with regular blending. Right: two sprites combined with additive blending.

achieve the desired effect: the function call

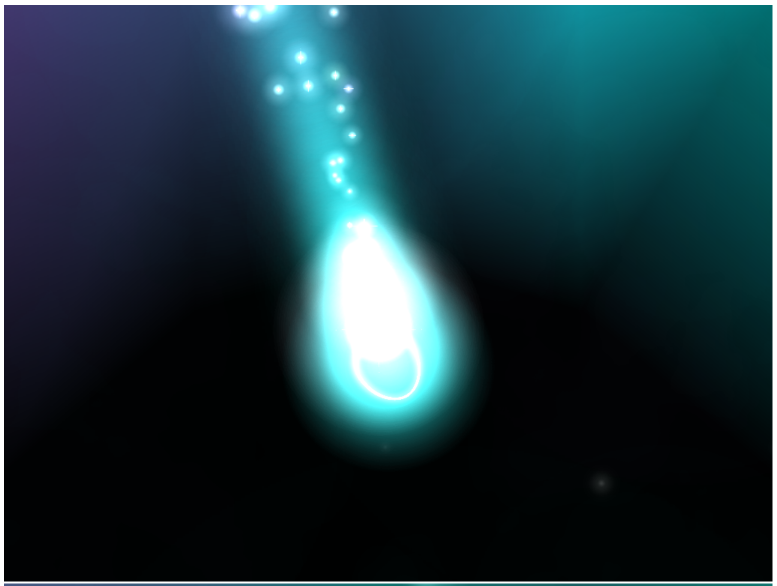
```
glBlendFunc(GL_SRC_ALPHA,GL_DST_ALPHA)
```

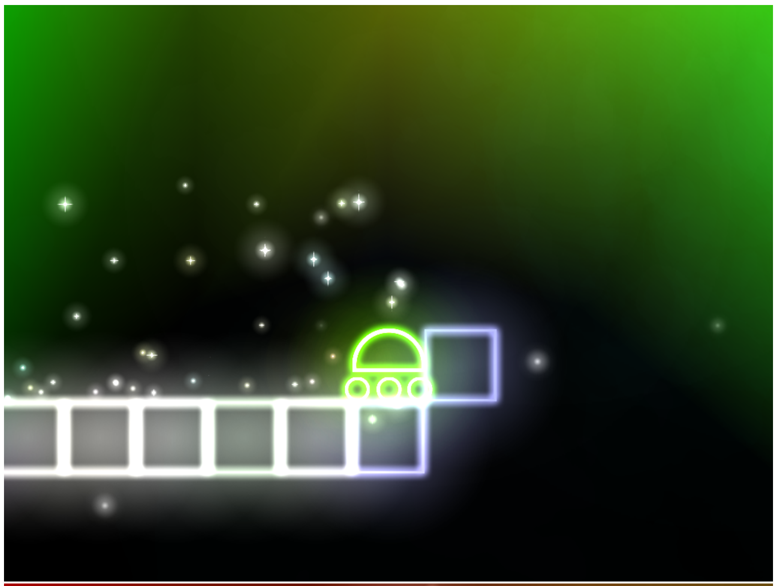
created the additive effect I wanted.

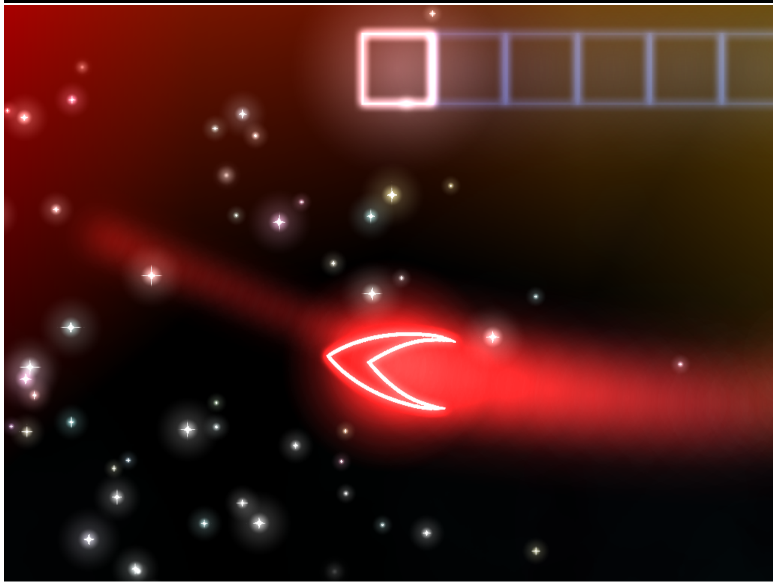
The next visual component to implement was glowing trails which would be left behind by moving entities. I found the most practical way to do this was to create an simple Entity-derived class, which would be used to represent a 'bloom entity'. This would be a non-colliding type of entity which would appear only as a faint bloom on the screen. The lifespan of a bloom entity would be short; its opacity would decrease with each passing frame until it would no longer display, whereupon it would be deleted from the scene. A function was created which would emit a trail of bloom entities in the wake of a moving object. The moving object's colour values would also be copied to each new bloom entity as they were created. This appeared to have the desired effect; however, the routine created a performance hit when applied to every object in the scene. Ultimately I applied the trail effect only to the player object.

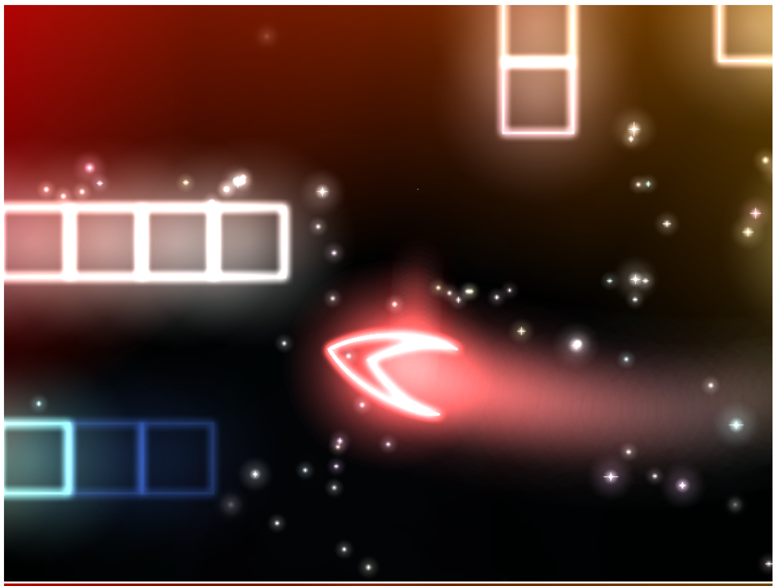
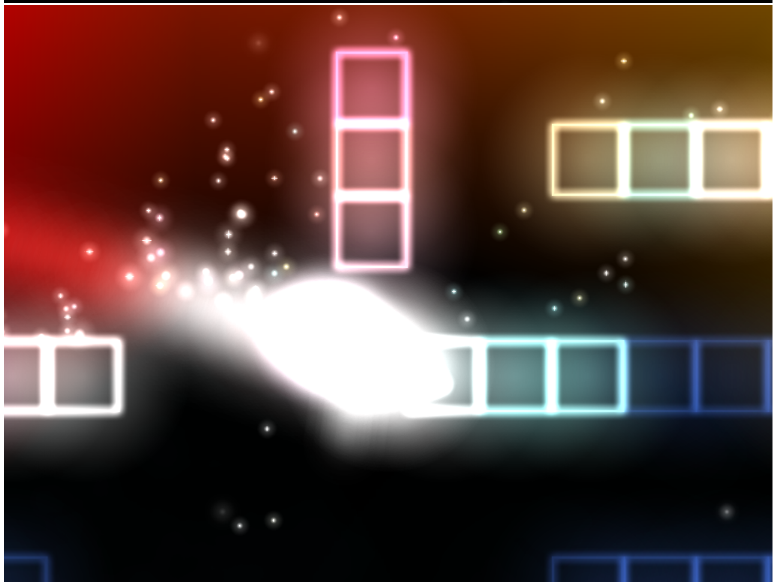
One significant problem I had with the visuals was the use of perspective projection. Although all game elements were to be drawn on a 2D plane, it was my intention to draw other decorative elements closer or further from the camera in order to create a sense of depth. The problem I encountered was a consequence of the wraparound stage design: the camera was not moving smoothly around a large space, but was instead jumping instantaneously from one stage boundary to another. The wraparound technique made this invisible on the 2D plain; however, objects placed at different distances from the camera would reveal the jump. I was forced to find an alternative method for creating a sense of depth; ultimately resorting to an orthographic view with false parallax applied manually to background and foreground elements.

The following images are examples of the final visual style I achieved.







5 Analysis

Achieving my design goals in this project was a very steep challenge for me. This was my first attempt to make a complete program in C++, and also my first encounter with object-orientated programming practices within the context of real program development. Broadly speaking, I feel that the project was a success; both in the sense that I achieved my stated goal, and also on the personal level, in that I significantly advanced my own skills over the course of the project. However as with any project, some aspects were less successful than others. Following is a discussion of the successes and failures of the project as I perceive them.

5.1 Successes

My stated aim was to create a program which provokes playful and exploratory interaction from users. In this sense I believe it was a success. In the late playtests I conducted during the game's development, users seemed compelled to continue experimenting with the game beyond the stage of initial curiosity. Continual visual feedback was a central component of my original design, intended to emphasize and encourage player interaction; the first sign which suggested the feedback was a success was when my friend and playtester insisted on colliding an in-game object against a wall repeatedly for several minutes. This behaviour lead me to believe that my ideas on interface design and visual feedback were at least partially correct.

Other user behaviours lead me to draw favourable conclusion about different aspect of the design. One reaction I noted during development was that when presented with an abstract image, users can fully misinterpret even the most minimalist of screen compositions. Some players were confused by the appearance of stars in the background of the composition; temporarily abandoning their progress in an attempt to provoke a response from the background elements. To me this helped confirm that a minimalist visual style was the correct choice for this project. A more detailed and elaborate style would likely have proved too confusing for players to interpret easily without instruction. This observation also argued in favour of the staged, sequential reveal of the game's visual elements.

The control system was perhaps the biggest unknown quantity when it came to implementing the game. I knew that in order for the controls to be intuitively accessable, they had to be immediately available on the mouse- precluding the use of keyboard input. I proposed several ideas for achieving a one-button control scheme in the design phase; however, it was possible that all of these ideas would prove to be uncomfortable or unsatisfying. The tactile quality of a control system cannot be established without hands-on testing. With this in mind, I had to implement several of my proposed control schemes before finding the most successful. The rotational movement combined with 'click-to-jump' proved to be easily the most tactile and entertaining of them all. The final implemented control scheme seems to work well with a mouse; and in addition, the control system becomes significantly more fluid and ergonomic when operated via a graphics tablet.

5.2 Shortcomings

One of the immediate shortcomings of this project is that its overall success is not easy to evaluate. I set out several goals when embarking on this project, but it is difficult to establish whether all or most of them have been met. The primary goal was to make a game which users should feel compelled to play to its conclusion, without any prompting or instruction. Unfortunately I did not allocate enough time to the playtesting stage to establish whether this is really the case. Initial responses were positive; but it would much require more thorough testing to discover if the project is a success in this regard.

Similarly, one of the main principles which informed the game's visual style is the ludic design idea of 'projection'. This is the idea that ambiguity can stimulate a richer and more imaginative response from users than literality[3]. It might be the case that the game's visual style has this effect on users; however, the scope of this project did not encompass a methodology for finding out. One way of doing this might have been to "re-skin" the game to a style closer to a real-world scenario. For example, the three stages of player evolution could be loosely refitted to represent the story of a bird escaping the nest and developing the ability to fly. This version of the game would then need to be tested with audiences alongside the abstract version, to establish which presentational style creates the most compelling experience.

Other more specific problems exist in my design. Some playtests have suggested that the opening section of the game is slightly *too* cryptic in its presentation; users would sometimes fail to discover the type of input needed to progress beyond the opening screen. The rotary mouse control scheme is unusual to the point that it simply does not become clear to some players. One way to rebalance this would be to include some more subtle visual cues in the opening stages - a spinning light or rotating object - together with more instantaneous feedback when the correct type of control is performed.

Another issue is the final "flying" stage of player evolution. This aspect of the game did not get as much development time as I had hoped for. The first problem with its implementation was the way in which the flying state is triggered; there seems to be an element of randomness to its appearance, which undermines its effectiveness as a reward for user input. Originally I planned to implement a more complicated set of conditions for triggering the flying stage; a chain of well-timed jumps would need to be executed sequentially at top speed, with the final correct jump launching the player into flight. This would have established a much stronger connection between the player's input and the appearance of the flying stage. Ultimately, there was not enough time to implement this feature.

A second problem with the flying stage is that the movement during this section seems to lack the physicality achieved elsewhere in the game. The ground-based movement system has a very pronounced sense of inertia, emphasized by the control system, which helps create the impression of a physical space. The flying stage lacks this sense of weight; it sweeps around instantly with no resistance to player input. An additional control challenge would probably have made this section a more engaging and satisfying experience.

On the technical side, I felt that my programming abilities were only just sufficient to complete this project. The program performs adequately, but the code used to achieve this result falls significantly short of efficiency. Collision detection was a particular area which I felt was poorly executed in this program. Part of the problem was my unfamiliarity with object orientation making it difficult for me to plan the program's structure in advance.

6 Conclusion

I believe this project has been very successful for me. Initial goals aside, the full implementation of the game in C++ and OpenGL has been a significant personal achievement. In addition to this, the resulting program does appear to achieve my initial goal; the feedback received from test audiences has been positive, and the game appears to engage users in the way I intended. Because of this I feel that I have managed to bear out, at least partially, some of my original ideas relating to game design and ludic design principles.

7 Bibliography

References

7.1 Game and Interface Design

[1] Bitt Animation and VFX, http://www.bittanimation.com/ [Accessed March 2008]

[2] Gaver, B., 2002. *Designing for Homo Ludens*, i3 Magazine No. 12

[3] Tarkaa, M, 2002. *Ludic Design - Notes from Bill Gaver's lecture at UIAH, Helsinki*, http://interactionmasters.uiah.fi/notes/billgaver.htm [Accessed March 2008]

[4] Tidwell, J. *Designing Interfaces*, http://designinginterfaces.com/ [Accessed March 2008]

[5] Sony Computer Entertainment 2006. *Locoroco*, released for the PSP handheld system

[6] Morrison, M., Mitchell, P., Brereton, M., 2007. *The Lens of Ludic Engagement: Evaluating Participation in Interactive Art Installations*, ACM Multimedia Interactive Arts Program

[7] Rouse, R., 2001. *Games on the Verge of a Nervous Breakdown: Emotional Content in Computer Games*, Computer Graphics Magazine

7.2 Programming Resources

[8] Burns, R. and Sheppard, M., *N Tutorials - Broad-Phase Collision*, http://www.harveycartel.org/metanet/ [Accessed March 2008]

[9] Andra, M. 2002. *GFX with SDL*, http://cone3d.gamedev.net/ [Accessed March 2008]

[10] *OpenGL FAQ*, http://opengl.org/ [Accessed March 2008]

[11] *World Coordinate Picking*, http://djfroofy.livejournal.com/3924.html [Accessed March 2008]

[12] *Linked List Class - C++*, http://www.dreamincode.net/code/snippet82.htm [Accessed March 2008]

[13] Shreiner, D., Woo, M., Neider, J., Davis, T., 2006. *OpenGL Programming Guide - Fifth Edition*