Table of Contents

1. Overview .................................................................................................................................................. 5
   1.1 Game Logline
   1.2 Game Synopsis
   1.3 Goals
      1.3.1 Intended Gameplay Experience
      1.3.2 The Minimum Viable Product
      1.3.3 Beyond the Minimum Viable Product

2. Audience, Platform, & Marketing ........................................................................................................... 8
   2.1 Target Audience
   2.2 Platform
      2.2.1 Why the iPhone?
      2.2.2 The App Store
   2.3 Marketing

3. User Interface .......................................................................................................................................... 10
   3.1. Heads-Up Display (HUD)
   3.2. Menu
      3.2.1 Title Screen
      3.2.2 Main Menu
   3.3 Control Scheme
      3.3.1 Movement Controls
      3.3.2 Camera Controls

4. Gameplay .................................................................................................................................................. 15
   4.1 Features of the Minimum Viable Product
   4.2 The Core Mechanic
   4.3 Colors
      4.3.1 Color Tiles
      4.3.2 Color Receptacles
      4.3.3 Color Mixing
   4.4 Obstacles
      4.4.1 Walls and Floors
      4.4.2 Color Removal Tiles
      4.4.3 Color Hold Blocks
      4.4.4 Spinning Tiles
      4.4.5 Pillars
      4.4.6 Elevators
   4.5 Abilities
      4.5.1 The Destroy Ability
      4.5.2 The Slide Ability
      4.5.3 The Spin Ability
      4.5.4 The Create Ability
      4.5.5 The Sticky Ability
      4.5.6 The Flip Ability
4.6 Features Beyond the Minimum Viable Product

4.6.1 Multiple Cubes
4.6.2 Gravity Blocks
4.6.3 Day and Night
4.6.4 Angled Platforms
4.6.5 Multiplayer

4.7 Level Progression

4.7.1 Completing a Level
4.7.2 Teaching the Player
4.7.3 Ordering of Levels
4.7.4 Progression of Difficulty
4.7.5 Finishing the Game

4.8 Playtesting

4.8.1 Usability Testing

5. Story

5.1 Synopsis
5.2 Story & Gameplay

6. Art Direction

6.1 3D Models
6.2 The Cube Texture
6.3 Level Textures
6.4 Level Backgrounds
6.5 Other Assets
6.6 Concept Art

7. Sound Direction

7.1 Background Music
7.2 Sound Effects

8. Technical Specifications

8.1 Technical Analysis

8.1.1 New Technology
8.1.2 Major Software Development Tasks
8.1.3 Challenges
8.1.4 Estimated Resources Required

8.2 Development Platform & Tools

8.2.1 Unity
8.2.2 The iPhone Emulator
8.2.3 Level Editor

8.3 Interface Development

8.3.1 Camera

8.4 Physics & Collision Detection

8.5 Art Development

8.5.1 3D Rendering
8.5.2 Implementing Level Backgrounds
8.5.3 Integration of Art Assets
8.6 Sound Development
  8.6.1 Integration of Sound Assets
8.7 The iPhone Application
8.8 Code Organization
  8.8.1 The UML Document
  8.8.2 Version Control
  8.8.3 Coding Style
8.9 Testing

9. Design History

9.1 Prototype
9.2 Color Cube

10. The Team

10.1 Team Members
10.2 Team Roles
  10.2.1 Design Lead / Product Manager
  10.2.2 Producer / Scrum Master
  10.2.3 Art Liaison
  10.2.4 Artist
  10.2.5 Sound Liaison
  10.2.6 Composer
  10.2.7 Level Designer
  10.2.8 Quality Assurance Lead
  10.2.9 Lead Engineer
  10.2.10 Engineer
1. Overview

1.1 Game Logline
Chroma is a 3D puzzle game for the iPhone that presents the player with color and movement based challenges which they must solve by mastering abilities and overcoming obstacles.

1.2 Game Synopsis
In Chroma, the player plays as a simple cube in a series of 3D levels filled with challenging puzzles. These puzzles are the fusion of color and movement based gameplay mechanics. The player is able to attach colors to any of the faces of their cube by first having that face make contact with a colored tile found in the level. The player can then move with that color attached to their cube, but if any face with a color on it touches the floor or wall then it is lost. The player must then pick up the color again. The ultimate goal of any level is for the player to transfer every color to its matching receptacle. Once this is done the level is solved and the player can progress to the next level. Levels are made more challenging through the use of abilities and the addition of new obstacles or through different combinations of familiar obstacles.
1.3 Goals

1.3.1 Intended Gameplay Experience

The gameplay we hope to achieve with Chroma adheres to the formula of “easy to learn, but difficult to master”. We want a game that mobile gamers can pick up and play, but can also find deep and challenging puzzles. The goal to achieving this combination is slowly introducing the player to all of Chroma’s mechanics by involving the mechanics in increasingly more difficult levels. By slowly teaching the player about the mechanics of our game by incorporating them in simpler levels first, the player will understand how they work and will not be caught off guard when they see them in more challenging levels.

However, though the player will become very familiar with all the mechanics of our game as they play, the true challenge comes from using many of these mechanics in a single level. Already knowing the characteristics of each mechanic, the player will need to understand the interactions between them so they can find solutions to the puzzles. Having many simple mechanics and then combining them will create a rich experience that is easily approachable yet deeply engaging due to the opportunity for players to have mechanics they have learned combined together to challenge them and give them an opportunity to master new puzzles.

1.3.2 The Minimum Viable Product

The Minimum Viable Product (MVP) is the game that our team has agreed is possible to make in the time period given to us for the CMPS170 sequence. The MVP only contains the gameplay features that our team feels are absolutely necessary to communicating our intended experience while still not providing a workload for us that is overly stressful and near-impossible to complete. The MVP is simple but sweet and will display our team’s ability to make a game that meets our goals and skill level as game developers. The features of the MVP for Chroma are all gameplay features described in section 4 except for those in section 4.6.
1.3.3 Beyond the Minimum Viable Product

Everything that is not in the MVP for Chroma falls into the beyond category. These are features that our team feels would make our game a more engaging but are not necessary to creating the play experience we want players to have. If we were to look at our development in an ideal way then, our MVP would be done by the end of CMPS171. This would mean CMPS172 could be used to polish the MVP and start adding features from our beyond MVP list. The beyond MVP features for Chroma would explore new directions to experience the core of our game and new ways to complement it. The features that are beyond MVP for Chroma can be found in section 4.6.
2. Audience, Platform, & Marketing

2.1 Target Audience

As we plan to develop Chroma for the iPhone platform, our target audience is primarily the mobile gamer. The average mobile gamer has been categorized as 28 years old, female (53% chance), North American (64% chance), has a household income of $66.1K, holds a bachelor’s degree (61% chance) and is white (66% chance). This audience prefers games with strong replay value, constantly increasing challenges and difficulty. The challenge that arises with this is that they also prefer to be able to play these games in short sessions. The puzzle game style of Chroma, with its focus in beautiful aesthetics, is the exact type of game this target audience would enjoy playing. Most of the users will likely be casual gamers, so we must be sure to create a game that is easy to pick up, while also adding challenges consistently as the player progresses. This audience is much different from the typical 18-34 hardcore audience that video games have been marketed to in the past, and we plan to keep our target audience in mind as we develop our product.

2.2 Platform

We’ve decided to develop Chroma for the mobile marketplace and with the limited resources we have to buy licenses for Unity, we decided to only purchase the iOS license for Unity and produce exclusively for iPhone users.

2.2.1 Why the iPhone?

We believe the iPhone is the best platform for Chroma because it provides us with a large user base, easy distribution, and a large amount of casual gamers suited for our style of game. The simplistic nature of touch control mechanics needed to generate a quality mobile game is well suited for Chroma’s game-play. Unity will allow us to easily port our game into the Mac App Store.

2.2.2 The App Store

The Mac App Store will be our primary means of distributing Chroma. The pricing structure will likely follow the traditional of having two versions of the game, one free and one paid. We will pay for the iOS developer license during winter quarter when we’re ready to begin testing on different hardware. We will also be mindful and abide by the list of rules and regulations regarding apps submitted to the App Store.
2.3 Marketing

Twitter, Facebook, Word of Mouth, Chroma Website

The mobile market is a very volatile and potentially rewarding market space in today’s gaming industry. As such it is very modern, innovative space which does not necessarily benefit as much from traditional means of marketing and advertising, such as bill boards, Internet banner advertisements or TV ads. We plan to use social networking sites, word of mouth and a personalized Chroma website and blog to spread excitement about our game.

Facebook and Twitter are unique broadcasting (aka. marketing) strategies in the social media space. Facebook is better at being a space for the general audience while Twitter provides more intimate connectivity with its users. Facebook’s pages allow users to respond to general announcements easily as well as asking questions. The reason Twitter provides a more intimate connection than Facebook is that it is designed to allow messages to come and go, while anything on Facebook tends to remain prominently displayed. This means the kinds of messages one can send on Twitter are different from the ones generally found on Facebook. As Quickdraw from Bulletproof Outlaws explains in “Indie Game Marketing: ARTICLE I – Social Marketing”, the power behind Twitter is that this intimate connection allows you to broadcast to a potentially very large audience:

“When you release or update your game, that’s 100 people Tweeting, Facebooking, etc. about your game’s news and now you’re reaching 10,000 people through them that you never would have had direct access to.”

Quickdraw explains that this personal connection is formed usually not necessarily by people being interested in your project, but of them being interested in you so it is very important to have personality.

Of course simple and direct word of mouth is also great way to garner personal interest in a project. Family, friends and colleagues will be most accessible in this way. A Chroma website is needed to give people a place to check out new developments without signing up for any services and to show a degree of professionalism.
3. User Interface

3.1. Heads-Up Display (HUD)

The HUD will consist of two translucent buttons. One will be the pause/main menu button and the other will be the camera control button. They will appear in the lower left and right of the screen. We will have a toggle option available to switch the side on which the buttons are located. This will make it convenient for use by people of both left and right-handed persuasions.

3.2. Menu

3.2.1 Title Screen

Once the player selects the Chroma game icon on their IOS product, the screen will dim and show the Chroma title. The Chroma title will await any sort of motion on the touchscreen before going into the main menu.
3.2.2 Main Menu

This main menu will be composed of a 3 by 3 plane interface with the controllable player cube in the center of it. Here the player will be allowed to move their tile among the menu options. From the player’s cube there will be four sets of options. North will be a square that leads to the main game and will have the text play floating above it. East will allow the player to view their best times of their completed levels, the text will display records. To the South will be the credits square that will show the names of the designers and engineers who worked on this project, the hovering text will read credits. Finally, to the West will be the settings option that allows the player to alter the volume of the sound effects and various other miscellaneous game effects, this will be labeled settings.
Now when the player selects the **play** option, their cube will be teleported to the overworld of the main game. This overworld will be composed of a 3 by 3 cube that the player can freely traverse. Each square will hold a single level, and each side will have a specific theme. The player need just double tap the screen to start whichever level tile they stand on. To return to the main menu, there will be a button on the bottom left corner of the screen that will read **menu**. Tapping this will return the player to the main menu. Completed levels will fill with a color corresponding to the overworld’s side. Each side has its own ‘completed level’ color similar to a Rubik’s cube.

![Image of Chroma game interface](image)

### 3.3 Control Scheme

#### 3.3.1 Movement Controls

In order to keep Chroma a fun and interesting experience, the player has to feel that the controls are comfortable. To get comfortable controls for Chroma on the mobile platform, we are going to keep it simple. The player may move the cube by swiping their finger in the desired direction. Just having this for movement, however, would cause frustration, because along long stretches the player would have to swipe their finger in the same direction an undesired number of times. To alleviate this, we will add in a functionality that allows the player to keep moving in the same direction if the player holds down their finger after swiping. For later puzzles that incorporate multiple cubes, the player may simply tap the cube they wish to use and it will switch over to that cube.
3.3.2 Camera Controls

Because Chroma is aimed for the iPhone, the user interface has to allow for all the required functionality, while it avoids obscuring the view of the player. Puzzle games such as Chroma need to have a decent view of the playing field, so the player can plan out moves and not rely solely on trial and error. Thus the user interface for Chroma has to be very concise. In order to achieve a concise UI, with all necessary functionality, we decided upon having the majority of the screen available to the player for viewing the playing field. In one lower corner, there will be a small button with a camera on it for the player to alter camera angles during play. In the other lower corner will be a small button for accessing a menu.

In order to move the camera around, the player can touch and hold the camera button in the lower-left hand corner, then they can move the camera around by sliding
another finger around the screen. The camera rotates with inverted controls with respect to the finger, but there will be an option to change it. As the camera rotates around the player’s cube, the directional controls adapt to the new camera angle. These simple controls cover the entirety of the game’s need, and remain easy enough to be comfortable for the player.
4. Gameplay

4.1 Features of the Minimum Viable Product

The following sections describes all the gameplay elements that are apart of the minimum viable product for Chroma. These are all elements that our team believes make up the core of our game and are therefore necessary to creating the intended gameplay experience we want players to have.

4.2 The Core Mechanic

Chroma has a very simple core mechanic: Place a color on your cube and transfer the color to its matching receptacle. The only problem the player has is that they cannot let a color touch the floor or the wall, otherwise they lose the color. All other gameplay elements are designed around this simple mechanic in order to make the act of transporting colors on the cube a more engaging experience.

4.3 Colors

There are six colors in the game that the player can interact with. They are: Red, Yellow, Blue, Orange, Purple, and Green.

4.3.1 Color Tiles

These are the tiles in the world that give color to the side of the cube that touches them. They cannot be destroyed or moved.

4.3.2 Color Receptacles

These tiles have a color associated with them. When a side of a cube with the correct color on it touches the receptacle, the corresponding colored puzzle is cleared and that color appears in the playing field and background.
4.3.3 Color Mixing

When one of the primary colors is added to a side of a cube that already has a primary color on it, the two colors are mixed on the cube and a secondary color is created. The combinations are: Red + Yellow = Orange, Red + Blue = Purple, and Yellow + Blue = Green.

4.4 Obstacles

Various obstacles are in each level. These obstacles either impede a cube’s progress through the level, can be used to navigate through the level, or can be used to assist a cube in solving puzzles.
4.4.1 Walls and Floors

These obstacles make up the playing field. The player must avoid touching the colored sides of their cubes to these tiles or the colors will be lost.

4.4.2 Color Removal Tiles

When a cube touches one of these dark tiles all color is removed from that cube.

4.4.3 Color Hold Blocks

When a colored side of a cube touches a side of one of these blocks, the block takes the color from the cube. When the cube touches the block again, it takes the color back. If both the cube and the block have a color, the colors will mix, if possible, otherwise they will swap.

4.4.4 Spinning Tiles

When a cube touches these tiles the are rotated 90 degrees, in place, in the direction indicated by the spinning arrows on the tile.
4.4.5 Pillars

These colored obstacles can be raised or lowered by fulfilling the corresponding colored receptacles in the level. These can be used to block or open paths and to build or destroy bridges.

4.4.6 Elevators

These obstacles are activated by fulfilling the colored receptacles of the same color. Elevators can move a cube by one tile when a cube touches the elevator tile.

4.5 Abilities

There is a special ‘ability’ tile on the playing field that behaves like a color tile (cubes pick up the ability ‘color’). When a cube mixes a color with the ability tile, the cube receives a special ability associated with the color they mixed. This ability is on a face of a cube and is activated by touching that face to a wall or floor tile. The six abilities are listed below.

4.5.1 The Destroy Ability

This ability is created from mixing with the color red. Destroy allows a cube to destroy a block on the playing field which makes for easier puzzle solving or allows access to new areas of a level.
4.5.2 The Slide Ability
This ability is created from mixing with the color blue. Slide allows the cube to move up to three tiles, without rolling. Sliding is a fast way of moving over a distance without changing what sides of a cube are colored.

4.5.3 The Spin Ability
This ability is created from mixing with the color yellow. Spin allows a cube to rotate 90 degrees, in place, in the direction the player chooses. This makes for easy colored cube side changing.

4.5.4 The Create Ability
This ability is created from mixing with the color green. Create allows the cube to make a new block in the world. This ability is activated by rolling a cube with the ability off of the edge of a platform. A new block is created below the cube.

4.5.5 The Sticky Ability
This ability is created from mixing with the color purple. The sticky ability grants the player the power to attach to another block creating a rectangular 3D block. This will allow the player to control this new rectangular shape and using it’s unique shape to navigate certain obstacles in the player’s path.

4.5.6 The Flip Ability
This ability is created from mixing with the color orange. When triggered all sides of the cube are swapped with their opposites. This can be advantageous in situations requiring the player to move a necessary pasted color from one side to the other.

4.6 Features Beyond the Minimum Viable Product
Up to this point all gameplay elements that have been described are apart of the Minimum Viable Product. There are other gameplay elements that we would like to incorporate into Chroma but might not have the time for. The following elements focus on new ways of approaching level design in Chroma as well as exploring modes of cooperation.

4.6.1 Multiple Cubes
A multiple cubes mechanic is just as the name suggests and would allow the player to play as more than one cube. This would be done by the player controlling one cube at a time. The player would control this cube just as they would a normal cube because it is a normal cube they are controlling. However, if the player touches the screen with three fingers they will instantly switch to another cube that is placed in the level. This new cube would then be under the players control just as the previous cube was. Each cube the player can control is a fully functional player agent in that each one has all the characteristic of a normal player cube. This means each cube can move around, collect colors, place colors on receptacles, and use abilities. Each cube can do
anything a normal cube can because all the player is doing is controlling multiple normal cubes. It’s as if the player is playing co-op with someone else only they are the someone else.

The goal of having multiple cubes is of course to create a situation of self cooperation for the player. Any cubes the player is not currently in control of would act as Color Hold Blocks only these blocks are not stationary and can be taken control of and moved around and used to solve puzzles and use abilities, as stated before. The element of self cooperation comes in because of this fact. There could be many situations where the player needs to swap or give colors to one of their cubes, take control of those cubes to set up a solution to a puzzle, and finally solve that puzzle.

4.6.2 Gravity Blocks

The standard level design for Chroma is that the player moves about on top of a level composed of ordinary blocks and is unable to go on the sides of these levels or on the bottom of them. The gravity blocks mechanic would allow the player to move on the sides and bottom of certain portions of levels similar to many levels in Super Mario Galaxy. Unlike ordinary blocks that are used to make levels, where the players can only move on top of them, gravity blocks would allow the player to move in all three dimensions instead of two. However, despite the new freedom of movement afforded to the player, gravity blocks would still act as normal blocks that construct the level in that they are walls and floors the player can lose their colors on. Gravity blocks would have a discernible visual difference to them that would allow the player to distinguish them apart form ordinary level construction blocks. With gravity blocks level design possibilities in Chroma would greatly expand. Levels that have sections of it that are made of gravity blocks would provide very interesting puzzle challenges.
4.6.3 Day and Night

The day and night mechanic is built upon the gravity blocks mechanic. Since the player is now able to go on the bottoms of levels an interesting concept comes into play. There would be certain levels that would have a day state and a night state depending on whether the player was on the top or the bottom of a level. With this new state change, brand new puzzles could be designed that take advantage of one state or the other. One idea that our team has briefly discussed was the idea of glow in the dark colors that only activate at night.

4.6.4 Angled Platforms

The standard design of levels in Chroma is that of multiple blocks forming a complete level. All adjacent blocks form 90 and 180 degree angles with each other. Adding angled platforms into levels would break this stable but rigid architecture and introduce a more complex but free-form level layout. Angled platforms would mean that levels would no longer be made entirely of blocks and would require other shapes such as wedges and triangular prisms to create seamless connections between level pieces. With angled platforms there would now be slopes and curved pathways in levels that would make for interesting level design. If a gravity element were added then levels could have spirals and curved structures in which the player could move on and find themselves upside-down or at other odd angles. It would be similar to level design in the Super Mario Galaxy games.
4.6.5 Multiplayer

Multiplayer could take two different roads: players could cooperate with each other or they could compete with each other. If it is cooperation then multiple players could move through levels together, solving puzzles along the way. Player interaction would be exactly like the multiple cubes mechanic only there is no longer a single player and their can now be more than one cube that is active at a time. Just as a single player would use multiple cubes to solve puzzles, multiple players would use the same techniques.

The other side of multiplayer, as mentioned before, is competition. It’s difficult to imagine competition between players in Chroma without some time element so time would play some part in player versus player gameplay. There could be a mode where players could steal colors from each other and each player’s goal would be to see how long they could hold the color before time runs out. However, since Chroma is a puzzle game, players competing to solve puzzles faster than their opponents would make more sense than gameplay based on precision with controls.

If multiplayer is added then there is the new challenge networking to deal with. We would no doubt use local networking, such as Bluetooth, that would allow players in close proximity to each other to player together.
4.7 Level Progression

- Progression through the level (MVP)
- Progression through the game levels
- Progression through the game levels (beyond MVP)

**Progression through the level (MVP)**

At the beginning player will be guided through the series of simple levels that will teach the player how to move the cube, pick up colors, etc.

Similarly to the games like cut the rope, player will be gradually introduced to the new features in the game. From simple to complex:

At first player will learn how to roll the cube. Then how to pick up and deposit colors and use other game features to solve the puzzles.

In addition, player will be learning basic controls while going through couple of first levels. There are going to be directions/animations on the screen on how to do the gestures to perform a desired action.

Additionally, as the player solved the puzzles in a level, areas of the level will change to a certain color. Once the player has solved all the level’s puzzles, the level will be fully painted. The camera will shift to a top-down view of the level and a picture will be revealed to the player.
Progression through the game levels (MVP)

For the MVP we will have several levels to solve and paint. Player will be playing simple levels first to learn game mechanics. As player solves and paints the levels, more complex puzzles and new mechanics will be introduced to the player.

Progression through the game levels (beyond MVP)

Similarly to the MVP level progressing, player will start with simple levels and build his/her skills as levels progress. However, levels will now form a story that will be told through coloring the levels. Now when player solves and colors the level, solved level becomes part of the story in the color book. New level and part of the story will be unlocked for the player to color.
4.7.1 Completing a Level

A player will successfully complete a level once all color receptacles in the level are matched with their respective color. As the player manages to fill receptacles, the color that was contained in the solved receptacle will begin to bleed out into the map creating a rough preview of what image the level may represent with thick, blotchy colors. Once all receptacles are matched, the map will be washed over by a higher resolution version of the level in full color, looking familiar but much more detailed.
4.7.2 Teaching the Player

The puzzles will introduce new mechanics to the player as they progress from world to world. Within each world, pictures created from levels will have similar themes or art styles and worlds will structured around introducing mechanics to the player and the most basic functionalities in the early stages of a world, and then progress to the more advanced puzzles involving that specific mechanic that was introduced, as well as any others that were introduced in previous worlds. This system of player education is heavily derived from similar level structures of games like Portal and Limbo.

4.7.3 Ordering of Levels

Levels will be ordered according to their difficulty as well as separated into separate worlds each emphasizing on certain new mechanics of the game. Levels in the early stages of a world will typically be easier than those later in the world. Each world will build upon the mechanics and powers introduced in the previous levels and use all of them together to create interested level design.

4.7.4 Progression of Difficulty

The difficulty will rise within each world and will drop back slightly as new mechanics and powers are introduced, however each new world will be slightly more difficult overall than the previous world. By the end of the game the player will be solving advanced puzzles that will be more difficult for the average user to complete.

4.7.5 Finishing the Game

The game will possibly conclude with some sort of scripted boss battle involving solving a level either within a time limit or something to that effect depending on where we decide to take the narrative, but for our MVP we will simply have a final level that will be the threshold for difficult level design we have at that point in time. We hope to finish the game in a way that allows us to expand upon it with further world releases.

4.8 Playtesting

For Chroma, we plan to utilize playtesting at nearly every stage of the design process. We will use this playtesting to gain insight into whether the game is achieving the player experience goals we would like or not. This will be a general overview of our plan for playtesting as it will likely evolve as we progress in the development of this game. During the early stages of development, with early prototypes, we will focus on self-testing the game. The reason for this is because at such an early stage the game is at a much more fragile state and the team itself offers the best help for proper feedback at this stage. At this early stage we plan to start getting a detailed record of feedback together, so that we can track the data and progress as we move forward.

Following this self-testing only phase of development, we will bring in friends and family outside of the team to assist with the playtesting. This is for building up the structure of the game. The reason for not going straight to external playtesters is that at
At this stage the game and controls will take a lot of explanation. It is important to test with people you are familiar with to better be able to communicate how the game works.

Once the game is complete enough where we can give much simpler instruction to the playtesters, we will start bringing in external people who fit within our target audience. This part of the playtesting is for when the game is at a point where we need feedback on formal details, and not so much on the structure of the game. For this phase of playtesting, we will try to get as large a variety of playtesters as possible, while also returning to the most articulate early testers for greater feedback.

For playtesting, we plan to design a one-on-one testing environment with a test script. The test script will be a guideline of things to tell the tester, and questions to ask. This test script will assist in the taking of detailed notes on the playtesters use of the game. After the playtesting session, we will have some sort of exit interview where we ask many of the questions we are most curious about at the time. With all of this data we will attempt to create user feedback reports that can be given out to the team to illustrate what the users think of the game as it is developed. This is extremely important, as we will always be able to get feedback from our target audience to make sure the game we are developing will be a successful one. It will keep up from straying too far from the important part of game development, creating a fun experience for the player.

4.8.1 Usability Testing

These playtesting sessions will not solely be for the purpose of gameplay feedback. As this is a game for a multi-touch device, we plan to combine usability testing into these sessions. This is important as we are defining our own control scheme, and the best way to create a solid control scheme is through usability testing. Our team has two members who are published for their work on multi-touch device usability, so we believe this will lead us to a successful control system and interface. We will combine the usability testing with the playtesting by including questions on how the user felt about the control scheme, and by allowing users to try multiple control schemes in a play session.
5. Story

5.1 Synopsis

Welcome to the planet Chroma! A vibrant world comprising of colorful landscapes and beautiful scenery...that is, until the ominous black cube devoured all of the planet’s hues leaving Chroma a lifeless gray rock in space. It is up to you, the player, to return this planet to it’s original beauty!

5.2 Story & Gameplay

The story of our game is revealed through the actions of the player. As the player solves puzzles in a level, a picture will be revealed to them. At the end of a level a complete picture will be shown and this picture will tell a portion of the story of our game. The storyboard below shows a very simple example of how the player would slowly reveal the picture in a level as they solve puzzles.
Storyboard Notes:

The Coloring Book levels will need to be expansive if made in the fashion of 3D pixel art.

- Possible Coloring pieces:
  - a Rubix Cube
  - a Heart
  - a Companion Cube
  - a Flower
  - a Mario Head
  - a Fruit Bowl

- Pro: Simple and fast to create
- Con: Really large amount of ground to cover per Level
6. Art Direction

6.1 3D Models
The models we will be using for Chroma are primitives, and thus do not require any special models. In the instance that we decide to improve upon the geometry of some objects, we have team members that are proficient in the use of Blender.

6.2 The Cube Texture
So as to not confuse the player, the main cube will remain mostly a simplistic white color. If time permits, we may improve the textures of the border of the cube.

6.3 Level Textures
Our level textures will be pleasant to the eye and will provide the player with distinctions for each specific tile. The textures for each tile will represent what type of tile it is, and will intend to give the player a sense for what the tile does. We will renovate the tile designs the we currently have to better emphasize the ability of each special tile. When a player finishes a level, all the tiles that have been colored through puzzling will dissolve into a full picture.

6.4 Level Backgrounds
We intend for the level backgrounds to be simple in design with vivid colors. The background colors are also going to be different from the puzzle colors. We want to give the player a relaxed feel, while avoiding a situation in which the colors of the background make the puzzles more difficult to solve.

6.5 Other Assets
We intend to have a very simple translucent menu button with some icon that represents a menu. When clicked/touched, it will bring up a menu for the player. The menu will likely be singular colored and translucent.
6.6 Concept Art
7. Sound Direction

7.1 Background Music

The background music will be focused on creating a calm tone that allows for users to focus on the puzzles and not be too distracted by the music, while also enjoying what they do hear. We are interested in taking the soundtrack in a more ambient tone with light retro-style jingles to accompany them. (For the main menu, we will have something that follows a similar style, but is catchy enough to get stuck in the player’s head.)

Music Examples: Bejeweled Deluxe 2, Marble Mania, Tetris

7.2 Sound Effects

We intend to keep the sound effects minimal to avoid breaking the calm atmosphere provided by the music, but we will have aural cues for each action and changing of state of the cube. For instance, there will be sounds for losing a color, gaining a color, rotating the cube, activating an ability, and clearing or activating an obstacle. There may be sound effects to accommodate important actions such as the loss of color.
8. Technical Specifications

Our platform of choice is Unity. Here are the specs of the software we will be using.

- Particle Systems
- Lightmapping
- Camera
- Terrain Engine
- Vertex Snapping
- Scene Editor
- Available importation from Maya, Cinema 4D, 3D Studio Max, Blender, etc.
- Physics add-ons such as Continuous Collision Detection which handles collisions of fast moving objects
- And many, many, more features are available in the Asset store.
- Some features may only be available in the Pro version such as Networking, Deferred Rendering, Occlusion Culling, a Debugger, etc.

8.1 Technical Analysis

Unity is free to a large proportion of developers and affordable for the rest, with the revenue being strong enough to keep Unity Technologies profitable and rapidly growing - this business model has revolutionized the games business. Unity is free to a large proportion of developers and affordable for the rest, with the revenue being strong enough to keep Unity Technologies profitable and rapidly growing – this business model has revolutionized the games business. It supports iPhone/iPod Touch/iPad, Mac, PC, Web, Wii, X-box 360, PlayStation 3, and Android. Over 700K registered developers including Bigpoint, Cartoon Network, Coca-Cola, Disney, Electronic Arts, LEGO, Microsoft, NASA, Nickelodeon, Ubisoft, US Army, Warner Bros., large and small studios, independent professionals, students and hobbyists. There are over 190k monthly active developers. (Source: unity3d.com)
8.1.1 New Technology

Unity Technologies biggest growth sector is in Android, since launching on it in 2010 (Unity Technologies has been on the iPhone since 2008; 10-20% of top selling games are iPhone games and more than 1500 Unity games are in the App Store today with many having made "best of" and "top selling" lists like Shadowgun, Battleheart, Gears, Samurai II and Snuggle Truck. (Source: unity3d.com)

8.1.2 Major Software Development Tasks

These are in no particular order:
- Implementing a skybox.
- Figuring out how to get basic art assets into Unity and using them.
- Implementing touch controls for the iOS.
- Implementing all of the power-ups for the Cube.
- A Level Editor for easy level making.
- Implementing an easy to use Camera that is user friendly with the player.
- Use lighting, physics, and shaders to make our game look awesome.

8.1.3 Challenges

- None of us have experience in developing for iOS, so figuring out how to implement touch controls and making it easy to use for this game may be a problem.
- Getting the team up to speed and familiar with Unity so that there aren't any problems of figuring out the interface.
- Working as a team in general may take a couple of weeks to get used to until we eventually start working efficiently.
8.1.4 Estimated Resources Required
- Unity Standard Edition – free of charge
- Mobile Platforms for Standard Edition. iOS at $400.
- Additional plug-ins for Unity may be required. The cost varies between $75 up to around $200.
- If looking into native code integration with social networks, multiplayer networking, or video playback, we need Unity Pro ($1500) plus iOS platforms which is it at $1500.

8.2 Development Platform & Tools

8.2.1 Unity
- Unity 3.0 Standard Edition
  - iOS support.
  - Free of charge with some minor spending in plug-ins when going for our MVP.

Our engine choice is Unity. Why? It is free, has mobile support, and is easily accessible by a majority of our team members who have experience in C#, C++, and Javascript. Though we may need to pay for some plug-ins and the platform kit that are required for our project, this cost is much cheaper in the long-term compared to what we may have to pay for other development kits (such as UDK). There are no royalty fees.

- Object Import (from Maya or 3DS Max)
- Beast Lightmapping
- Nvidia PhysX
- Plug-in support for things such as EZ GUI.

8.2.2 The iPhone Emulator
Unity 1.7 includes support for the iPad Simulator and universal applications so apps can be designed to work for iPad, iPhone, and iPod touch.

8.2.3 Level Editor
UnityGUI allows you to integrate your own unique level design tools, AI control tools, debugging tools, difficulty tuning tools, or anything else that you might need. There are over 130 new API entry points that enable you to create specialized, customized editor tools and build them into the existing Editor interface. The editor offers excellent facilities for inspecting and changing object properties in a running game session, but there’s no line-by-line script debugging.
8.3 Interface Development

8.3.1 Camera

Unity has its own built-in camera engine which allows you to have different Rendering Paths. For Chroma, we are probably going to have a static overhead camera with full view of the field. We are also probably going to make a skybox, and fortunately, Unity has support for this as it can add a skybox component to the camera. There are also the additional standard editions to the camera such as near and far planes, orthographic, and perspective camera views.
Perspective Camera

Orthographic Camera
8.4 Physics & Collision Detection

Unity has NVIDIA PhysX physics engine built-in. On an object, you may add a Rigidbody. What this does is it makes the object that you have specified in the engine affected by gravity which will also let it collide with other objects. Basically it lets your Game Objects act under the control of physics. The Rigidbody can receive forces and torque to make your objects move in a realistic way.

Colliders are another kind of component that must be added alongside the Rigidbody in order to allow collisions to occur. If two Rigidbodies bump into each other, the physics engine will not calculate a collision unless both objects also have a Collider attached. Collider-less Rigidbodies will simply pass through each other during physics simulation. (Source: unity3d.com)
8.5 Art Development

8.5.1 3D Rendering

Unity supports different Rendering Paths. Different rendering paths have different features and performance characteristics that mostly affect Lights and Shadows.

Forward is a shader-based rendering path. It supports per-pixel lighting (including normal maps & light Cookies) and realtime shadows from one directional light. In the default settings, a small number of the brightest lights are rendered in per-pixel lighting mode. The rest of the lights are calculated at object vertices.

Shaders in Unity are written in a language called ShaderLab which allow you to have your own custom vertex and fragment programs.

*Surface Shader without Texture*
8.5.2 Implementing Level Backgrounds

Skyboxes are a wrapper around your entire scene that display the vast beyond of your world and we are going to have them in our level backgrounds. The art for the skyboxes may vary level to level. You can add them to an individual Camera (usually the main Camera) or you can set up a default skybox in Render Settings's Skybox Material property.
8.5.3 Integration of Art Assets

For importing files, Unity can use two types of files: Saved 3D application files and Exported 3D formats. Unity can import, through conversion: Max, Maya, Blender, Cinema4D, Modo, Lightwave & cheetah3D files, e.g. .MAX, .MB, .MA etc.

Settings for materials in your native package will not all be imported to unity:

- Diffuse Colour, Diffuse texture and Names are usually supported
- Shader model, specular, normal, other secondary textures and substance material settings will not be recognised/imported. However, there are ways around this. You can simply create materials inside Unity itself. Since we are going for a basic design, it shouldn’t be too difficult.
- Store your output texture files from your Unity project into \Assets\textures, and then import.
- Unity already takes care of compression for the output platform, so the source should probably be in lossless format for maximum quality.

(An example of Art Assets imported into Unity shown above with descriptive names for easy finding)
8.6 Sound Development

8.6.1 Integration of Sound Assets
Importing sounds works just about the same as importing art. All you really need to do is find them, import them, and then select custom settings.

- Drag and drop sound files into the Project window. Making a special folder for audio is recommended for organization sake. Note: Unity will accept a large variety of audio files such as mp1, mp2, mp3, ogg vorbis, wav, and aiff along with audio tracker files such as mod files, IT, S3M, XM, and FMOD files.
- After importing sounds, clicking on any of them will bring up an Audio Importer on the Inspector window. In the Inspector window, the first choice will be Audio Format, where you can choose to either compress the sound file or leave it as it is. Generally, it's a good idea to have longer files, such as dialog, be compressed and have the shorter files be uncompressed.
- You'll have access to a Compression slider to control the bit rate of a compressed file. The lower the bit rate of a file is, the smaller the dynamic range and general fidelity will be. On the up side, though, lowering the bit rate does make for more memory to us.

8.7 The iPhone Application

General submission steps
To get our content to the Apple App Store involves three main steps:
1. Make our game!
2. Once we have our application built and ready to submit, we'll be using two sections of the iOS Dev Center. The first site is the Provisioning Portal, with which we should already be familiar from when we created our development certificates and provisioning profiles.
3. After we have the correct certificates and provisioning profiles, we will then use the second section, iTunes Connect, along with the Apple Application Loader utility, to prepare our app and submit it to Apple.

Getting our app ready
Before we start to package our app for submission, we will make sure to take care of some key considerations to ensure your users have the best experience possible. These considerations fall generally into two categories: performance and usability. To make sure we have built a high-performing app, we will test our application on a variety of devices (iPhone 3GS, 4, 4S, iPod touch).

Performance
Typically, the iPod touch is faster than an iPhone in the same class of hardware. For example, the iPod touch 32GB and iPod touch 64GB are faster than the iPhone.
3GS, even though they're in the same class of hardware. The iPhone 4 and the iPad both share a common processor, which are the fastest among all the iOS devices.

**Images and icons**

Apple requires that our application have an application icon, App Store icon, and launch image:

- **Application icons** should be supplied as PNG files in the following sizes to support the currently available iOS devices: 57 × 57 pixels, 72 × 72 pixels, and 114 × 114 pixels. One additional size, 29 × 29 pixels, is used for Spotlight search results on the device, and is optional.

- **App Store icons** are 512 × 512 pixels. For development builds, this icon is included in the application package and is referenced in the publish settings or application descriptor file. However, we do not include it as part of the application for the final release. The App Store icon for the final release is uploaded separately using the iTunes Connect website.

- **Launch images** are included in the root directory of the application package but are not referenced in the publish settings or application descriptor. iOS loads the correct image based on the filename. The name will be a variation of Default.png. iOS chooses the variant depending on the device and how the application is invoked.

  Once we've prepared our iPhone Application (.ipa) file, we'll need to fill out the required forms on the iTunes Connect site to submit the application to Apple.

**Other resources:**


### 8.8 Code Organization

#### 8.8.1 The UML Document

The Unified Modeling Language (UML) document will be the visual representation of the code architecture for Chroma. The purpose of this document is to map out the high level organization of the code and to see the major parts of functionality of our game and how they interact with each other. More specifically, we will be making a UML class diagram which, as the name suggests, shows a class level view of the code. We will also be making a UML sequence diagram which will show the major execution paths through our code. This is an important diagram because it will help us to see what pieces of our code architecture are run when different actions occur in the game. Furthermore, it would allow us to have an overall look at the order in which pieces of functionality are executed.
8.8.2 Version Control

A version control system such as SVN or Git will help the development process in a big way and is pretty much necessary for our team for multiple reasons:

- Our team will not always be able to work in the same physical space and a version control system allows for code consolidation regardless of distance between team members
- Just as the name suggests a version control system keeps multiple versions of committed code. This feature would allow us to move back to a previous version of our game if our current one became corrupted or compromised in some way
- A version control system keeps track of who is committing what, which is very helpful. Without a version control system it would be difficult to tell who programmed what and when.

Unity has a plugin to integrate SVN. We will look into possibly using it for our project:

8.8.3 Coding Style

Before development starts it is important for the team to agree upon a coding style to use. A coding standard for the team will ensure that code is readable and well organized. However, a coding style does not just dictate how the code should look, it also dictates how it should be coded in the first place. A classic coding style is the code-and-fix method. It is a style where programmers constantly make minor changes to the code and then run the code to see what happens. It is usually employed when a programmer is inexperienced or the architecture of their software is not fully known. Having a well thought-out UML document will help us stray away from the code-and-fix style.

8.9 Testing

We will be developing our game using Agile development model. Agile development model employs test driven development (TDD). TDD relies on the repetition of a very short development cycle: first we will write a failing automated test case that defines a desired improvement or new function, then produce code to pass that test and finally refactor the new code to acceptable standards.

We will use TDD to make sure that our game:
1. meets the requirements of our design and development;
2. works as expected;
Test-driven development cycle:

Add a test
In test-driven development, each new feature begins with writing a test. This test must inevitably fail because it is written before the feature has been implemented. (If it does not fail, then either the proposed “new” feature already exists or the test is defective.) To write a test, we must clearly understand the feature’s specification and requirements. We can accomplish this through user stories that cover the requirements and exception conditions. This could also imply a variant, or modification of an existing test. It will make us focus on the requirements before writing the code.

Run all tests and see if the new one fails
This will make sure that tests are working correctly and that the new test does not mistakenly pass without requiring any new code. This step also tests the test itself, in the negative: it rules out the possibility that the new test will always pass, and therefore be worthless. The new test should also fail for the expected reason.

Write some code
In this step we will write some code that will cause the test to pass. The new code written at this stage will not be perfect and may, for example, pass the test in an inelegant way. It is important that we focus on writing the code that is only designed to pass the test. That is acceptable because later we will improve, polish and hone it.
Run the automated tests and see them succeed
If all our test cases now pass, we can be confident that the code meets all the tested requirements. This is a good point from which we can begin the final step of the cycle.

Refactor code
Now we can clean up the code as necessary. By re-running the test cases, we can be confident that code is not damaging any existing functionality. There is going to be removing duplication code. We will be removing any duplication code between the test code and the production code — for example string that were repeated in both, in order to make the test pass in while writing the code to pass the test.

Repeat
Starting with another new test, the cycle is then repeated to push forward the functionality. We will be using small steps 1 to 10 edits between each test run. If new code does not rapidly satisfy a new test, or other tests fail unexpectedly, we will revert to excessive debugging. We will also be using continuous integration which will help with revertible checkpoints.

Testing levels
We will use following methods to test our game code:

Unit testing
Unit testing or component testing, involves tests that verify the functionality of a specific section of code, usually at the function level. These types of tests will be written as we work on code, to ensure that the specific function is working as expected.

Integration testing
We will use integration to verify the interfaces between components against a software design. Integration testing will help us to to expose defects in the interfaces and interaction between integrated modules.

System testing
System testing will allow us to test a completely integrated game to verify that it meets its requirement.

System integration testing
We will use integration testing to verify that our game is integrated to any external or third-party systems we will use in the game requirements.

Regression testing
Regression testing will help us with finding defects after a major code change. It will help to to uncover old bugs that have come back. We might encounter bugs after
integration or code change, but before the change our game functionality was working as intended. We will test our code by re-running previously run tests and checking whether previously fixed faults have re-emerged.

Acceptance testing
Our acceptance tests will consist of the following:
1. An acceptance test prior to introducing a new integration or major code change.
2. Acceptance test performed by the game testers to make sure our game functionality meets our and payer's expectations.
9. Design History

9.1 Prototype

Our prototypes helped us to discover how players approach the puzzles in the game, what controls are intuitive to players, where players get stuck, and how the handle getting stuck. Our focus is to be a casual game, so minimizing frustration is very important. Several design decisions for this came out of prototyping, such as never telling the player that they are wrong or displaying a lose screen. The prototypes also revealed that art is very important for our game. Some playtesters found it difficult to differentiate between certain tiles, so we locked down our art style for the future iterations. Prototyping helped us to get a feel for the level design of the game, get to know what puzzles are difficult for players, and to get to know the difficulty curve we are going to have to have for the MVP.

Doing digital prototypes made us tackle some coding issues early on, such as the relationship between cube movement and camera control. The cube must move intuitively whenever the user presses a move key, no matter how the camera is oriented; pressing 'right' always moves the cube to the right, in the view. This was something we didn’t think about early on, but had to address for the sanity of the player.

9.2 Color Cube

The original idea and most of the first prototype came from an idea that we had in 2008. We revisited this idea in the beginning of our CS 170 sequence to improve the idea and make it bigger and better. We then discovered that in 2010, someone released a game (Color Cube) that is incredibly similar to our prototype. Color Cube has nearly identical textures, puzzles, and gameplay to our early Chroma prototypes. We have taken this discovery and used it as motivation to continue our build-up of our original idea. We now have many features already implemented in our prototypes that Color Cube doesn’t have and have expanded on ideas that are in both games to keep our design separate and innovative.
10. The Team

10.1 Team Members

**Team Name**: Spectagon Studios

- Eric Wieber
- Christopher Burris
- Wai Son Wong
- Grey Skold
- Reid Weber
- Vadim Maximov
- Harsha Potiny
- Joshua Harris
- Scott Orzech
- Yarlinda Jiang
- Petar Bojinov

10.2 Team Roles

10.2.1 Design Lead / Product Manager

**Role Held By**: Eric Wieber

It is the responsibility of the design lead to handle all design related matters of game development. Their goal is to preserve a clear design vision that is proposed by the team as a whole, but ultimately finalized by the design lead themselves. They should take suggestions from team members and outside advisers to refine and implement the design of the game. They should work closely with the level designer and the quality assurance lead to refine levels, the art liaison to refine the art style, the sound liaison to refine the sound direction, the lead engineer to ensure a code implementation that supports the design, and finally the producer to ensure as many design features make it into the game as possible.
10.2.2 Producer / Scrum Master
Role Held By: Christopher Burris

It is the responsibility of the producer to ensure a game development process that is as efficient and smooth as possible for the team. Scrum is a major tool the producer, and the team as a whole, uses to keep track of tasks being done and progress being made. Above all, the producer is a facilitator. The producer must make sure that the team is working as effectively as possible by enforcing a strong line of communication between all team members, helping to resolve team member disputes, and by keeping tasks clearly defined and completed by the agreed upon deadline. The producer works closely with the design lead to ensure a practical set of features that can be developed given the allotted time and resources for development. They also work closely with the art and sound liaisons to keep artists and composers as much apart of the team as the engineers.

10.2.3 Art Liaison
Role Held By: Wai Son Wong

It is the responsibility of the art liaison to enforce a strong line of communication between the artists and the rest of the team. They are effectively a producer that focuses primarily on facilitating all matters pertaining to the art direction of the game. The art liaison makes sure that the desired art style of the design lead and other team members is communicated clearly to the artists. However, the art liaison also makes sure that the ideas and needs of the artists are taken into account and taken as seriously as ideas and needs from other team members. The art liaison should meet with all the artists on a regular basis to ensure artists are completing their tasks, their needs are met, and they share the same artistic vision as the rest of the team.

10.2.4 Artist
Role Held By: Yarlinda Jiang

It is the responsibility of an artist to communicate with the art liaison and the rest of the team in order to decide upon an art style that they will then create. Artists should understand the design of the game so their artwork effectively complements and supports the gameplay and design vision. They should have a strong line of communication with the art liaison to ensure their ideas and needs are being heard and seriously considered by the other team members. Artists should meet with the art liaison regularly to show progress on tasks they're working on and to stay up-to-date with any team related news. Finally, artists should work with the art liaison and engineers to ensure their artwork is correctly represented within the game.
10.2.5 Sound Liaison

**Role Held By:** Grey Skold and Reid Weber

It is the responsibility of the sound liaison to enforce a strong line of communication between the composers and the rest of the team. They are effectively a producer that focuses primarily on facilitating all matters pertaining to the sound direction of the game. The sound liaison makes sure that the desired sound direction of the design lead and other team members is communicated clearly to the composers. However, the sound liaison also makes sure that the ideas and needs of the composers are taken into account and taken as seriously as ideas and needs from other team members. The sound liaison should meet with all the composers on a regular basis to ensure composers are completing their tasks, their needs are met, and they share the same musical vision as the rest of the team.

10.2.6 Composer

**Role Held By:** Petar Bojinov

It is the responsibility of a composer to communicate with the sound liaison and the rest of the team in order to decide on a sound direction that they will then create. Composers should understand the design of the game so their sound assets effectively complement and support the gameplay and design vision. They should have a strong line of communication with the sound liaison to ensure their ideas and needs are being heard and seriously considered by the other team members. Composers should meet with the sound liaison regularly to show progress on tasks they’re working on and to stay up-to-date with any team related news. Finally, composers should work with the sound liaison and engineers to ensure their sound assets are correctly represented within the game.

10.2.7 Level Designer

**Role Held By:** Grey Skold and Reid Weber

It is the responsibility of the level designer to ensure that the desired gameplay experience of the team is communicated effectively to the player and that all gameplay mechanics form a balanced and engaging experience. The level designer should work closely with the design lead and the quality assurance lead to make sure all gameplay elements are effective in creating the desired gameplay experience. Level progression is a necessary element the level designer must consider when designing levels and the interactions of mechanics that occur within them. They should design the sections of the game so the player learns clearly all actions available to them and what outcomes those actions have. The level design should challenge the player but also leave them with a sense of mastery after overcoming those challenges, thus keeping them engaged.
10.2.8 Quality Assurance Lead

Role Held By: Scott Orzech

It is the responsibility of the quality assurance lead to organize playtest sessions, gather feedback from playtesters, and clearly communicate that feedback to the team to ensure potential changes are made and effective game elements are reinforced. The quality assurance lead should work closely with the design lead and level designer to relay and document information from playtesters. The quality assurance lead should also lead the rest of the team in effective playtesting procedures to ensure that feedback is clearly understood and taken as corrective criticism and not insults. The quality assurance lead should employ multiple ways of playtesting to effectively get feedback from playtesters. Above all, observation is the most important skill of the quality assurance lead.

10.2.9 Lead Engineer

Role Held By: Vadim Maximov

It is the responsibility of the lead engineer to help form and maintain an organized and effective code architecture for the game. They are also responsible for working with the engineers of the team to effectively test the code of the game to ensure it supports the desired behavior of all game elements. The lead engineer should work closely with the design lead and level designer to make sure all desired gameplay features are being implemented correctly. The UML document is a large undertaking for the lead designer and they must work with the engineers to create it and to ensure the implementation of all game elements is clear to all the engineers. The lead engineer should also work with the engineers to form an agreed upon coding style to make sure code is readable and well documented. Lastly, the lead engineer should work with the art liaison, sound liaison, and engineers to ensure that art and sound assets are properly integrated into the code architecture.

10.2.10 Engineer

Role Held By: Harsha Potiny, Joshua Harris, Eric Wieber, Christopher Burris, Wai Son Wong, Grey Skold, Reid Weber, Vadim Maximov, and Scott Orzech

It is the responsibility of an engineer to understand the code architecture of the game and to help implement and test it in an effective manner. They should work with the lead engineer to create the UML document and to pick sections of the code architecture they will implement and test. Engineers should have a clear understanding of the design of the game so that the code can effectively support it. They also must work with team members working on art and sound assets in order to integrate them into the code architecture. Engineers should hold themselves to an agreed upon coding style to make their code readable and well documented so that other engineers can...
understand it. Finally, engineers should use a version control system to ensure code organization, backup versions of files, and easier merging of code.