[[1]](#footnote-1)

Creating a Competitive Multiplayer Open-Arena 2D Twin-Stick Shooter

Anthony Cloudy and Squirrel Eiserloh

*Abstract*— This thesis intends to demonstrate an all-around mastery of the lessons and skills developed through the Guildhall’s software development track by building a competitive, open-arena, two-dimensional twin-stick shooter from the ground up. The artifact iterates on *Kirby Air Ride*’s “City Trial” by creating a more competitive and progression-focused game. *AllStar* demonstrates a holistic understanding of video game programming, including gameplay systems, engine coding, shaders and graphics programming, all while emphasizing polish.

*Index Terms*— Game Development, Real-time Rendering, Software performance, Software quality

# INTRODUCTION

Mastery of programming for game development requires a vast knowledge base covering a multitude of difficult and varied skills. While much of the software development industry requires more niche roles with less cross-pollination of disciplines, game engine programmers must understand everything from input and real-time rendering to advanced data structures and networking. This thesis intends to demonstrate a mastery of the lessons and skills developed through the Guildhall’s software development track by building a competitive multiplayer, open-arena, two-dimensional twin-stick shooter from the ground up. The project aims to demonstrate an all-around mastery by creating a well-polished game, titled *AllStar*, in a custom C++ engine.

The artifact applies optimization techniques to create a 60fps gameplay experience for up to four players, even while they roam the map in different areas. The game pushed its engine’s codebase to its limits, and acts as a demonstration of what’s possible in the author’s engine built from his Guildhall experience. The game features split-screen multiplayer that pits players against one another in an arms race to build the most powerful ship in a set time. Players explore an open arena, destroying cargo crates and enemy ships to earn upgrades to their ships. Upgrades affect the base stats of a player’s ship, enabling her to go faster, tank more damage, or shoot more powerfully. A player can hunt others in this game mode to steal some of her opponent’s resources, as all player’s stats and equipment are frozen once time is up. The player then must use her powered-up machine in three randomly-chosen contests: including but not limited to a battle royale, a race, or a coin-grabbing challenge. Because the competition is picked randomly, a player has no idea what kinds of minigames she’s going to compete in after the Assembly phase, which adds to frantic and fast-paced nature of the game.

The artifact was built using a public GitHub repository to log all code commit messages, which helped discern what was done when and kept progress transparent. A development diary was kept not only to document what challenges arose during development, but to also document decisions and problems resolved during the artifact’s creation. Concentric development separated the game’s feature development into tiers, defining clear stages for the project and creating milestones.

# Research Review

Because of the competitive nature of the thesis, this literature review focuses primarily on finding resources on competitive game design and creating multiplayer experiences. This also includes research into specific challenges the artifact faces, including split-screen game design and shoot-em-up (*shmup*) design. Games that demonstrate either a strong competitive/multiplayer design are of equal importance to the research, and provide proven examples of what works and what does not. Other games are in the research review for their specific shmup qualities or control styles that provide reference for the artifact’s design. The researcher consulted Professor Squirrel Eiserloh and Professor Christopher Forseth from The Guildhall at Southern Methodist University to find games with mechanics, playstyle, and gameplay similar to the proposed mastery project. The researcher then studied on their suggested games, searching Steam with keywords including *Galak-Z* and *Realm of the Mad God*. The researcher expanded off of those initial results, using the “More Like This” section to find even more related games. The researcher also utilized his personal games library to find more similar games, such as *Kirby Air Ride*. Finally, the researcher utilized Bing and the SMU Central Libraries search with keywords such as “Competitive Games”, “Competitive Game Design”, and “Splitscreen Game Design” to find articles and research various aspects of gameplay the artifact will utilize. The researcher limited search results and games only to those localized in English.

## Literature Review

In "Rock Paper Scissors - A Method for Competitive Game Play Design", author Victor Chelaru discusses the nature of Rock Paper Scissors, or RPS design in games, where certain attacks have an absolute advantage or tie with others (just like the game the design’s namesake shares). The article goes in-depth on the metagame of “Pure RPS”, where the attacks have no lead up or predictability (grounded units vs flying units in an RTS), “RPS and Signals”, where attacks do have readability (such as the wind-up animation of a punch), and “RPS with separate Attacks and Signals”, where attacks have signals, but experienced players can cancel or feint signals. The article reveals the dominant, winning strategies for the above games that emerge from gameplay, and discusses ways to keep the game from incentivizing the wrong kind of gameplay. Because some of the dominant strategies that evolve include “be random and fast” and “don’t initiate any attacks” for the more basic RPS designs, ignoring the insight this article has could destroy the metagame for any competitive multiplayer game. The dominant strategy that evolves from the most advanced RPS design is to adapt to your opponent’s patterns, which will develop a healthy, competitive game that prioritizes player skill and reading your opponent without promoting stale tactics [1].

The article on “Shared-Multi-Split Screen Design” by Richard Terrell assesses and compares the distinctive design considerations and limitations provided by various types of multiplayer screen designs. The article exposes some of the tradeoffs and design challenges that split-screen games face. Split-screen gameplay can force a reduction in graphical quality, as the game must render two to four separate views every frame. The reduced screen space also can cause problems for players, as this space conveys important location information. Other design hurdles the article mentions include the introduction of screen-peeking, a need for increased monitor size to prevent feeling constrained, and increased team communication if players want to cooperate. The article fails to mention any positive aspects of split-screen as opposed to multiple screen, which include cheaper setups, greater flexibility when playing with other people, no network latency, and the potential for more positive experiences that come from playing with others in person [2].

The postmortem for *Good Robot* provides valuable insight into some of the unique design challenges shmups face. The developer, Shamus Young, started the project as a solo project, but eventually transitioned to work with another studio once he realized that his game’s design had issues. The postmortem outlines how he managed to resolve the game’s flaws by working with the other team’s ideas, which included establishing a dynamic gameplay rhythm, with valleys and peaks of activity, and adding consequence to player death. Many of Young’s concerns are pitfalls this thesis could encounter while in development, especially in regards to game design and mechanics not panning out or a lack of proper pacing in the game. Failing to give players the sense of enjoyable tension, or failing to create meaningful and interesting player interactions can jeopardize similar projects [3].

A paper titled “Group Report: Progression Systems” from Project Horseshoe 2014 deconstructs the nature of progression systems. The report broke progression systems down into a series of building blocks that make up system fundamentals, as well as tactics to strengthen player motivation towards interacting with the systems. Of the system building blocks described, the most relevant to the thesis include the Progression loops, which spiral upward as the players gain power in order to accomplish new feats that grant them new powers. The power up system, in which players continually make incremental progress on their ships, matches a power loop, where playing the game improves the player’s avatar’s power, which improves their “virtual skill” for the round. The paper also links player motivations, such as superiority and control, to rewards like competition and power, via “progression atoms”. Progression atoms are in-game components that serve as the conversion from the player’s motivations into rewards. By giving a player who wants better control of their character a set of character stats, they can give the player the reward of power through those stats [4].

## Field Review



Figure 1: Kirby Air Ride's City Trial mode features power-ups scattered all over the level that alter the characteristics of player’s machines [5]

*Kirby Air Ride* is a multiplayer, 3D racing game created for the Nintendo GameCube, which put players head to head while piloting a variety of quirky “Air Ride Machines”. The game features an alternate game mode called “City Trial”, in which players were put in an open map and given free roam for 5 minutes. Players began on a basic, neutral Air Ride, and are tasked with finding a better machine and collecting power-ups to customize their machine within in the time limit. At the end of the game, all players compete in a random minigame that tests the player’s skill and powered-up ride, with the winner of the competition winning the whole game. Although Terrel’s paper describes many of the design limitations of split-screen, *Kirby Air Ride* manages to utilize split-screen successfully to create an enjoyable experience despite the limitations, and many of the performance tradeoffs are either hidden by the game’s design or minor incidents (such as a few occasional framerate hiccups). The artifact for the thesis draws heavily upon City Trial’s gameplay for inspiration, and aims to push the boundaries of this original idea and take it to a new level. This thesis aims to utilize *Kirby Air Ride*’s unique gameplay style that provides randomness without arbitrary outcomes, while improving upon the game’s minimal player interactions and unwieldy combat [5].



Figure 2: Players battling monsters in Realm of the Mad God have to pay extremely close attention to their surroundings, as bullets come in various speeds and patterns that can end players lives instantly [6]

A fantasy bullet-hell with fast leveling and *permadeath* (the game deletes a player’s character when they die), *Realm of the Mad God* is an unconventional massively multiplayer online game (MMO). Players are thrust onto an open world where they travel to defeat enemies, gain experience, and loot corpses until all major bosses on the map have been vanquished. Once all the players have defeated the bosses, the whole server is thrust into a battle with the game’s final boss. The game is a twin-stick shooter, where players avoid bullets while desperately trying to land shots on the hordes of enemies on the screen. *Realm of the Mad God’s* map and player versus enemy (PvE) combat line up a significant amount with the design of the thesis artifact. Whereas players are incentivized to defeat enemies through the chance of rare equipment upgrades in the MMO, the artifact aims to use the power-up system to incrementally boost the player’s stats. The artifact ends up giving out a multitude of small power ups with a few and far-between large slot items, instead of a constant stream of items you may or may not be able to use. The artifact also aims to explore a similar control scheme to *Realm of the Mad God*, in order to give the players more control quicker without having to worry about being unable to “pilot” a physics-based ship. *Realm of the Mad God*’s pickup and equipment system is also important to the thesis, as players are inundated with a steady supply of weapons, armor and potions at a rate that matches the quick-paced nature of the game [6].

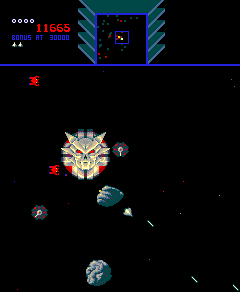


Figure 3: A player avoiding Sinistar while trying to create sinibombs. The game’s open arena and obstacles match the thesis’ design [7]

*Sinistar* is a top-down, multi-directional shooter where the player is locked in an arms race against “Sinistar”, the game’s villain. While workers attempt to reconstruct Sinistar, the player attempts to survive gunfire and mine planetoids to create “Sinibombs”, the only weapon that can defeat Sinistar. Once Sinistar is created, the player either needs to destroy him or run away, as getting caught by Sinistar results in instant death for the player. *Sinistar* provides an example of a PvE RPS balance that shifts over time, as it requires players to balance mining, direct attacks, and evasive maneuvers in order to win against Sinistar [1]. The player has to mine for sinibombs in order to attack Sinistar, but the act of mining leaves the player open to attacks from warriors. Without sinibombs, players can only evade Sinistar, as his attack trumps the player’s standard laser. The player’s options change in value before and after Sinistar is activated, creating gameplay dynamics that change over the course of the play session. *Sinistar* has very similar theming, handling, enemies and obstacles that the proposed artifact will contain. The idea of shooting level obstacles in order to acquire resources, the way the player navigates through the level, and the feeling of PvE combat in Sinistar match the thesis’ ideas. However, while *Sinistar* generates tension via PvE, the artifact will generate this tension mostly via PvP, as the arms race is between players, not against an almighty boss [7].



Figure 4: Galak-Z's unique handling and polish set it apart from other titles in the genre, creating the feel of actually driving a spaceship [8]

*Galak-Z* is a Shoot ‘em up, 80’s Sci-fi Anime styled Roguelike that casts players as a lone pilot fighting against enemies in cavernous planetary dungeons. The gameplay combines Roguelike gameplay with Shoot ‘em up controls to create a unique experience, as players pilot a physics-based rocket through various “dungeon rooms”. The game also values stealth, as the player’s rockets make noise that alert enemies to the player’s presence. The game’s unique aesthetic and levels of polish are high quality, and while mostly out of scope for the constraints of the thesis, serve as a great reference to aspire and work towards. *Galak-Z*’s ship controls are also intuitive, and the artifact aims to explore the physics-based nature of the ship to see if that would improve gameplay over *Realm of the Mad God*’s control scheme [8].

The majority of the games listed have some sort of RPS gameplay, as outlined by Chelaru’s paper [1]. *Kirby Air Ride* features jousting-based combat that utilizes RPS with separate attacks and signals, as players must approach one another to attack, and can easily feint an approach to sway their opponent’s behavior [5]. *Kirby Air Ride*’s constant stream of power ups grants the players more control, but the game fails to provide progression systems for other common competitive player motivations [4]. The flow and rhythm concerns that arose during the development of *Good Robot* are an obstacle some of these games overcome as well [3]. Although extremely hectic, *Realm of the Mad God* manages to establish this rhythm through the spacing of enemies in dungeons, and by giving the players the ability to break out of tight situations via instant teleport to a hub world [6]. Because players are able to lure and stack multiple enemies to create hordes that would obliterate the game’s flow via incredibly intense moments, giving the player the option to take a break at any point prevents the game from becoming overwhelming [6]. *Galak-Z* comes from the other end of the spectrum, where the majority of the gameplay isn’t hectic, but tension and flow is generated through stealth and using level obstacles to alleviate pressure.

## Summary

This artifact aims to create interesting competitive multiplayer gameplay while attempting to avoid the various pitfalls and issues discovered through research. By utilizing RPS with separate Attacks and Signals as a foundation for designing player’s options and interactions, the artifact can avoid stale or boring dominant strategies [1]. Without the separation of attack and signal, the best course of action becomes never initiating attacks, which detriments the game [1]. Although the thesis intends to be competitive, the players should not always be at each other’s throats, as mentioned in the postmortem for *Good Robot* [3]. This thesis attempts to establish a good gameplay rhythm by balancing player interaction with the map’s scale, allowing players the choice to fight and the space to run off and recover, without making the map too large for players to find one another. While *Kirby Air Ride* creates an interesting play space and encourages moments of interaction through gameplay events, the game fails to incentivize combat enough. Players must be extremely close to one another to consistently battle, and with the scale of the map and handling of the machines, the game fails to deliver an incredible PvP experience [5]. This project intends to play with and combine *Realm of the Mad God* and *Galak-Z*’s differing control styles to create the best combat experience for the artifact. As mentioned in Terrell’s article, split-screen has a host of downsides and technical limitations that the artifact has to overcome [2]. Considering that optimization is a part of the mastery the thesis wants to demonstrate, the project attempts to ensure that the game runs well even with 4 players on screen.

# Methodology

The artifact is designed to demonstrate the author’s mastery of the teachings and concepts taught in SMU Guildhall’s programming track. The project demonstrates gameplay, graphics, and engine programming, as well as the ability to procedurally generate content, optimize, and polish a game through code.

## The Game

*AllStar* is a competitive, open-arena, two-dimensional twin-stick shooter that runs in a custom C++ and OpenGL game engine. The game is for two to four players and is controlled using one to four Xbox/XInput controllers. A player moves with the left stick, and aims and shoots with the right stick.

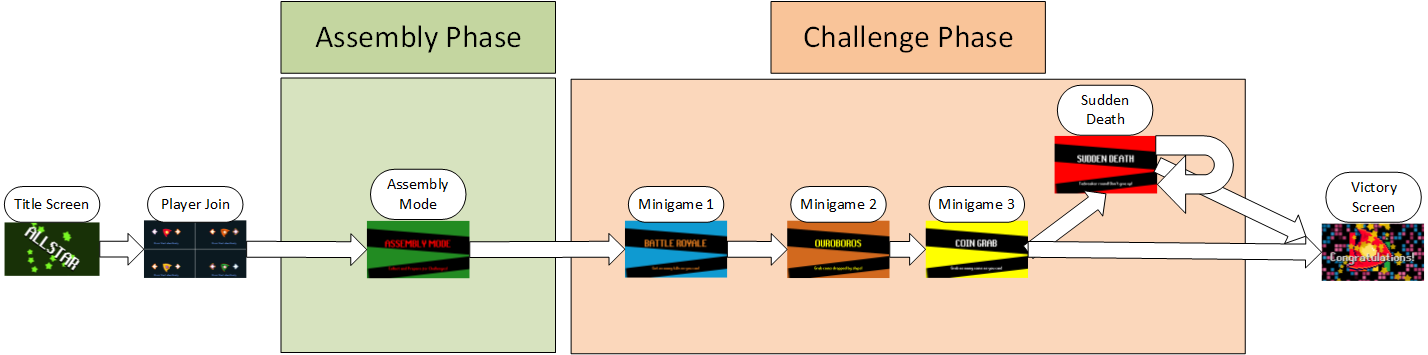


Figure : A game of AllStar, featuring the two main phases of gameplay, Assembly and Challenge.

A game of *AllStar* lasts a total of 10-15 minutes. A player starts on the Player Join screen, where she can pick her ship color and ready up for the game. Once all players are ready, gameplay goes through two phases, Assembly and Challenge. In the Assembly phase, a player flies around an open arena and scavenges for upgrades to her ship. In the Challenge phase, the player takes her upgraded ship and plays against her opponents through a trio of minigames to fight for victory. After finishing the final minigame and viewing the game’s winner, a player is returned to the title screen where they are given the option to play again and try out new strategies and combinations of upgrades.

At the start of the Assembly phase, a player starts with a default ship with no stat modifications, and is given five minutes to assemble her ship. The player roams an open arena, trying to find as many *power-ups* and *equipment* as she can to build a ship that suits her style. Power-ups are pickups that modify a player’s stats, while equipment are pickups that change the player’s abilities, weapon, and base stats. The specific combination of these pickups make up a player’s *build*, showing which of the player’s skills are augmented by the equipment and power-ups, and describes how the player’s ship is most likely to fare in different minigames.

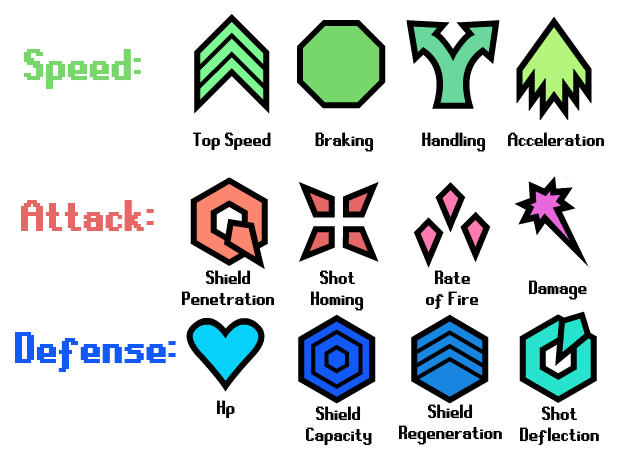


Figure : The twelve power-ups, in their respective power families

Picking up one of the power-ups increments one of the player’s twelve passive skills, such as top speed or shield regeneration rate. The power-ups are grouped into one of three families: speed, attack, or defense. Each family has four power-ups that affect the player’s stats in a similar manner. The speed family includes *top speed*, which increases a player’s maximum velocity, *braking*, which decreases the amount of time for a player to come to a complete stop, *handling*, which improves the time it takes for the player to change their direction of motion, and *acceleration*, which improves how quickly a player gets up to her maximum velocity. The attack family includes *shield penetration*, which increases a player’s damage bonus when attacking shields, *shot homing*, which increases the rate at which shots home in on enemies, *rate of fire*, which decreases the cooldown time between each shot, and finally *damage*, which increases the amount of damage each projectile does to other entities. The defense family includes *Hp*, which increases a player’s maximum health, *Shield Capacity*, which increases a player’s maximum shield health, *Shield Regeneration*, which increases the rate of regeneration of shield health, and *Shot Deflection*, which increases the rate at which shots are pushed away from the player as they approach. Each player is trying to get as many of these power-ups as they possibly can, as each one improves their stats for the rest of the game. A player can find power-ups by breaking boxes, destroying asteroids, or defeating non-player enemies.

Players are also on the lookout for *equipment*, which are pickups that provide a player with new abilities, weapons, and temporary stat bonuses that stack on top of a player’s 12 stats. Each player has four swappable equipment slots, one for each type of equipment. The four types of equipment are the *active*, a special ability that a player activates using the left trigger, *weapon*, the projectiles the player fires when shooting, *passive*, a gameplay-modifying bonus or always-on ability, and *chassis*, which is the player’s ship body. A player can find equipment by destroying any of the boxes scattered around the map, which has a chance of containing any of the above equipment.

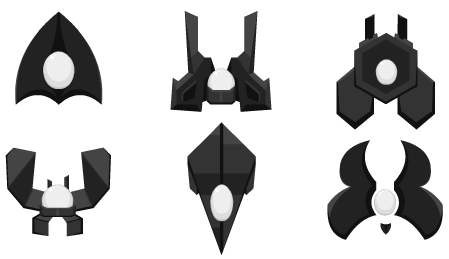


Figure : Concept art for the different chassis in the game.

The chassis is the foundation of a player’s build, as it has the biggest skill bonuses and drawbacks of all equipment. For example, the speed chassis greatly improves the player’s top speed and acceleration, but reduces handling and damage greatly, allowing the player to move quickly in straight paths, but turn slowly and widely. Each weapon shoots different types of projectiles, and have a large impact on how the player approaches enemies and other players. Certain weapons like the missile launcher encourage area control, while others like the wave gun encourage precision and proper spacing, causing different optimal strategies for each player’s build.



Figure : The spreadshot weapon (right) encourages players to get in as close as possible to blast enemies with as many shots as possible. The wave gun (left) outranges the spreadshot, but encourages the player to keep enemies at the focal point for optimal damage

Passives are equipment that provide a gameplay change passively, such as a player being able to cloak whenever her ship isn’t moving. Picking up the Spray and Pray passive encourages more area coverage with projectiles at the cost of reduced damage per projectile. Actives are activatable equipment that give the player a new ability that gives them a temporary edge over other players. These abilities range from temporary power-up boosts, like Quickshot’s large burst in rate of fire for 5 seconds, to more custom actions, such as Boost’s ability to dash forward in the direction the player is pointing, changing momentum and doing damage on contact with other entities.

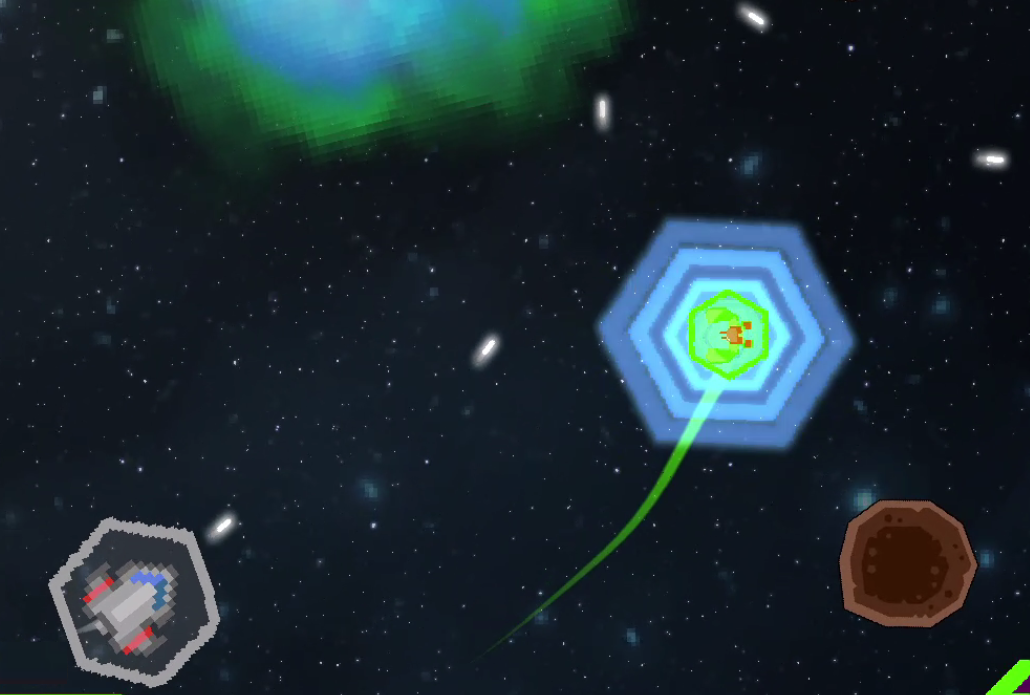


Figure 9: A player using their active ability to drastically increase their shot deflection.

The Assembly arena is filled with major and minor *encounters*, which are procedurally generated landmarks, enemies, and features that populate the world. Players can seek out encounters that let them customize their builds towards different goals, such as enemies or crates that drop speed power ups. A player can also stumble across environment landmarks, such as black holes and wormholes that create interesting strategical advantages and disadvantages. If a player dies during the Assembly phase, her chassis is destroyed. The dead player also drops a percentage of her power-ups, and potentially one of her non-chassis equipment pieces, that can be picked up by other players.



Figure : A player taking advantage of her environment to sneak up on an enemy ship.

After Assembly ends, the players are shown their stats, showing them the results of what they gathered. The players are then locked in to those power-ups and equipment, and can’t lose or gain any more for the rest of the game. After this menu, the Challenge phase begins, and the players must compete with their newly assembled ships for victory.

The Challenge phase of gameplay consists of a set of three mini-games. The mini-games challenge the players in a variety of ways, from straight up battling to racing, to survival challenges. Each mini-game round lasts two minutes, and players earn points based on their ranking. The player with the most points at the end of the three rounds wins!

AllStar 
No Respawns. Get 85 many kills 85 you can! 

Figure : One of the minigame splash screens

## Program Structure

*AllStar* utilizes object-oriented programming to add object interactions quickly and easily. The game’s primary orchestrator is *TheGame* class, which manages the various gamemodes and ferries the players across the modes during a game. A *GameMode* is an abstract superclass of the assembly mode and the various minigame modes. Each of the GameModes handles the creation of the world, initialization of props and entities inside the world, and keeps them all within bounds while updating the camera and other gameplay elements. TheGame handles all the transfer between GameModes, along with the results screens, while deferring to each GameMode to handle all the gameplay and player updating logic for the specific mode.

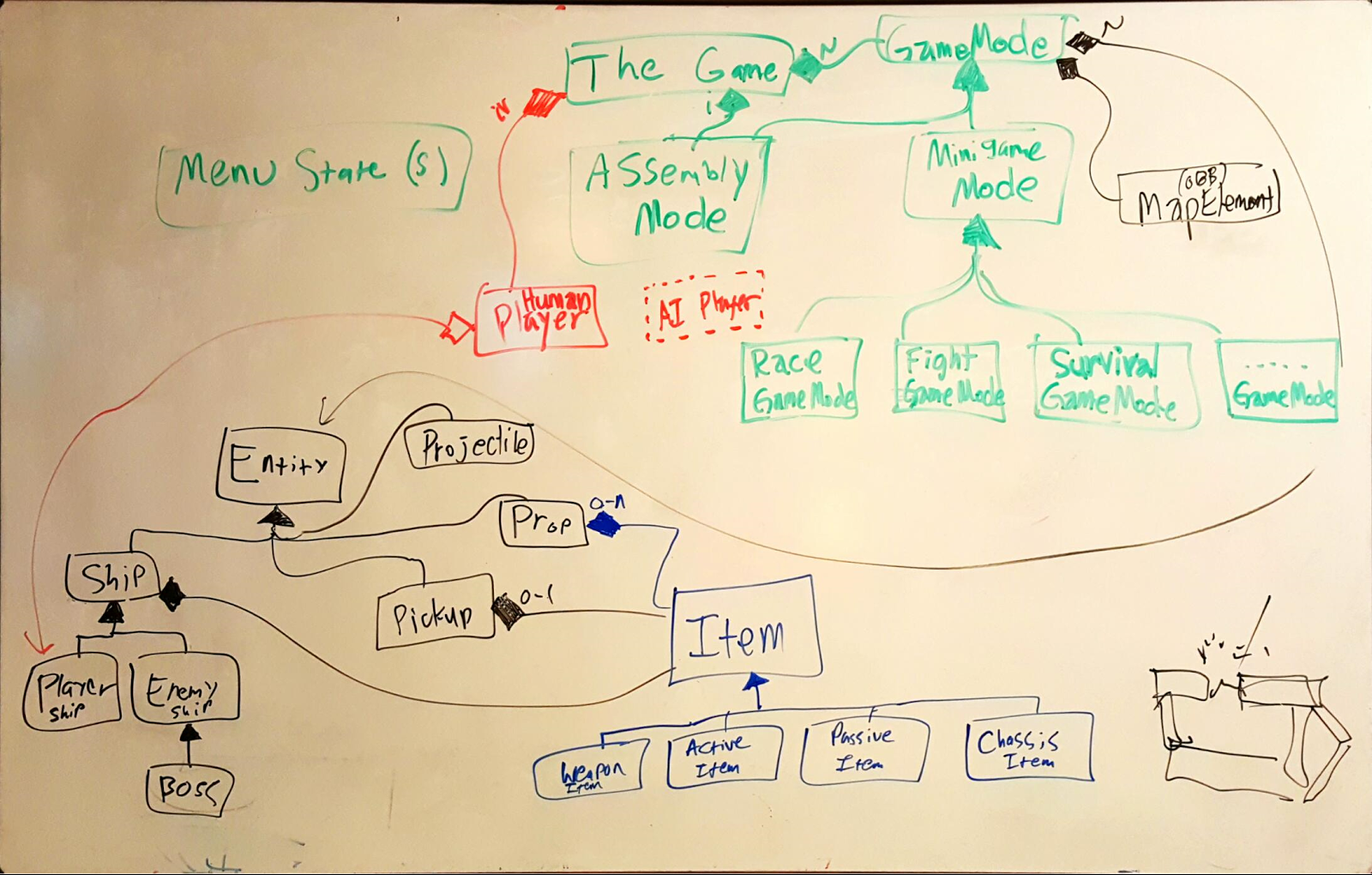


Figure : A UML Diagram that displays the core architecture of the program.

A GameMode handles the procedural generation of the game world by populating the world with props, enemies, and the players. Things spawned in the world are a subclass of the *Entity* class, which contains the common elements for game objects. All Entities have a sprite, are registered with the sprite renderer, and are drawn automatically. [Talk about the Sprite Game Renderer?] Each of the GameModes handles and updates all the Entities in the game every frame.

[Should I talk about entities here?]

[Pilots vs Ships?]

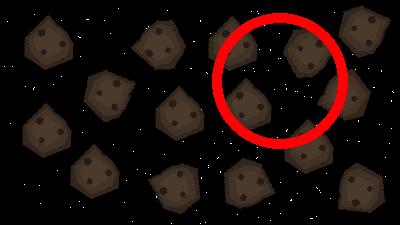
The game utilizes a procedural generation system to create each of the levels for the game. [Insert reason why this was done over tailored maps?] The process starts by adding anywhere from 50 to 100 asteroids to the map by randomly picking spots inside the arena.   
  


Figure : Stage 1- The map is filled randomly with many asteroids. The GameMode then selects a random circle in the map that doesn't collide with any of the other encounters.

After filling the map with asteroids, the game determines a set number of *encounters*, or map features, to spawn in the game. Each game mode picks how many encounters are in the map, and can control the amount and types of encounters it spawns. The game splits encounters into two groups, *minor* and *major* encounters, based on the physical size and magnitude of the encounter. For example, as a nebula is less gameplay-impacting and more passive, it is a minor encounter and spawned more frequently. Conversely, wormholes and black holes take up much more gameplay space and actively impact how players play the game on a much larger scale, and are thus large encounters and limited in the number of spawns they have in the world.

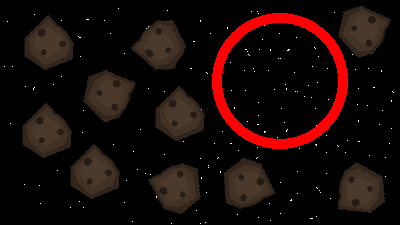


Figure : Stage 2- The GameMode clears out any entities within the radius of the encounter, which removes any asteroids that would be colliding with this new encounter.

After selecting an encounter, the game generates a random radius and attempts to spawn the encounter into the game. The game fills out the large encounters first, then moves on to the medium ones.

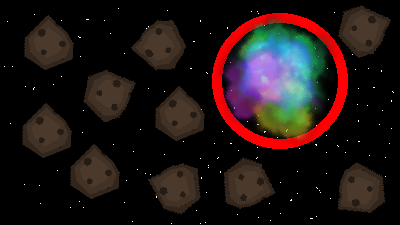


Figure : Stage 3- The encounter is spawned in.

Once the GameMode has selected a valid location for the encounter that doesn’t collide with any other encounters, the GameMode deletes any entities within the proposed encounter’s radius. The process checks for collisions with any of the asteroids in the game map, and removes anything that could potentially interfere with the encounter.

Finally, once the area is cleared, the GameMode spawns in the encounter. Each encounter object is coded using relative coordinates, which allows the entities within an encounter to be placed in a regular pattern based on the scale of the radius the cleared-out space.

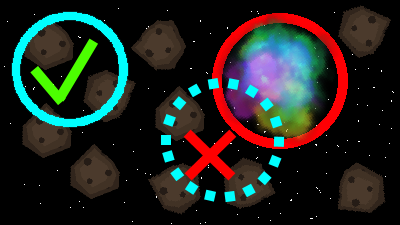


Figure : A new encounter attempting to spawn in collides with a previous and fails. A second attempt is made that collides with no others, and succeeds.

Subsequent encounters are spawned in checking against all of the previous encounters’ boundaries. This step is to ensure that no entities of another encounter are removed when clearing space for a new encounter. In the figure above, an encounter’s random location is too close to our previous encounter, forcing the encounter to pick another location.

## Development Process and Structure

[Should I include any of this?]

The project employed concentric development to organize the game’s components and features into discrete tiers. Each of the tiers consisted of a set of features that built off the previous tier’s work, and provided a clear path for the project’s dependencies. The tiers also provided priorities for the sets of features, and naturally divided up the features into milestones. The last tier was considered optional, and served as the stretch goals of the project.

Tier 0 consists of any mandatory engine work that needs to be done before beginning the project. Although the majority of the engine’s subsystems are up to the quality needed for the project, a few bugs with rendering and particle systems need to be addressed before continuing on to the remainder of the project.

Tier 1 consists of all the core gameplay elements that make up the game. These features focus on getting the game functional first, proving out the core loop and the gameplay elements before moving on to polish tasks. This tier includes implementing multiplayer, programming player ships and rudimentary enemies, the game’s basic power ups, and a level to fly around in. The game will also have two distinct modes, assembly (the main game) and challenge (the minigames segment), along with start and end UI, including the victory screen. Most of the content will not be polished to final quality, but instead be the foundation for the rest of the game’s features and polish to build off of. This tier is much akin to a Proof of Concept Gameplay milestone, but will be first playable as well.

Tier 2 contains tasks with a focus on getting the core gameplay smooth, polished, and feeling good. This tier is to mitigate the risk of overscoping up front and running out of polish time, so that before any new non-core features and functionality are added, the game already feels good. This will establish the minimum-viable product for the game, and will ensure that the project meets the goal of creating a complete and polished game.

Tier 3’s tasks focus on augmenting the current gameplay to improve gameplay quality and replayability. This tier introduces socket items/equipment, and adds the remaining power-up items and stats. Completion of the tier’s tasks will also add 2-6 more challenge variations, as well as procedurally generated map zones during assembly phase. These tasks will be polished to final quality, in order to match the quality of the game after tier 2 has been finished. Once tier 3 is completed, the developer should be able to halt development at any point and the game should still feel like a complete and polished product, ready for defense.

All the remaining tasks and stretch goals reside in tier 4, which is optional for completion. This tier includes tasks such as implementing bosses during assembly phase and adding more challenges. The tier will also add new slot items to the game, such as new weapons and chassis types. This tier will continue to push the bar of quality and polish for the game, and will be worked on during any of the remaining time in the project.

By using concentric development, the project is not only organized into discrete milestones with clear objectives and deliverables, but is separated into a chain of dependencies that prioritize the core components of the project.

Stat GrowthCurves 
din 
L 2 E 7 E LO LL LS LE LE LE L? 20 2L 22 25 2E 27 u 2? 

Figure : The potential stat growth curves for a stat. The green bars highlight the minimum and maximum power-up levels.

Game balance was one of the biggest struggles throughout the project. To reduce the difficulty of balance and expose how the game’s power-up stats worked and depended on one another, a spreadsheet was created to iterate on and test different power-up values. *AllStar’s* power-up skills range from 1 to 36 internally and from -5 to 30 externally. Instead of limiting the stats to linear growth, which wasn’t working well for the beginning of the project, the stats have an option of multiple curves to create a better growth trajectory. Figure 11 above shows the potential stat growth curves that each of the skills could follow. [Talk about debates over whether skills are linear, geometric, and how we exposed them to the player and reached these decisions?] Most of the skills ended up following the Smooth Stop trajectory, but a few implemented Smooth Start to keep the major effects from revealing themselves too early on. The leftmost green bar (level 6) demonstrates where players would start with a fresh character, and the rightmost green bar (level 26) demonstrates what power the player would be at after collecting the maximum number of power-ups for that stat (20 power-ups).

|  |  |
| --- | --- |
| **Stat Level** | **Top Speed** |
| 1 | 2.00 |
| 2 | 2.03 |
| 3 | 2.12 |
| 4 | 2.27 |
| 5 | 2.47 |
| **6** | **2.72** |
| 7 | 3.02 |
| 8 | 3.35 |
| 9 | 3.73 |
| 10 | 4.14 |
| 11 | 4.58 |
| 12 | 5.05 |
| 13 | 5.54 |
| 14 | 6.05 |
| 15 | 6.58 |
| 16 | 7.12 |
| 17 | 7.67 |
| 18 | 8.22 |
| 19 | 8.78 |
| 20 | 9.33 |
| 21 | 9.88 |
| 22 | 10.42 |
| 23 | 10.95 |
| 24 | 11.46 |
| 25 | 11.95 |
| **26** | **12.42** |
| 27 | 12.86 |
| 28 | 13.27 |
| 29 | 13.65 |
| 30 | 13.98 |
| 31 | 14.28 |
| 32 | 14.53 |
| 33 | 14.73 |
| 34 | 14.88 |
| 35 | 14.97 |
| 36 | 15.00 |
|  |  |
| **MIN** | 2 |
| **MAX** | 15 |

Figure : A row from the stat table that demonstrates the growth of a stat's value based on the stat's level.

Figure 12 above is a subsection of the table that calculates out and displays a stat’s growth based on skill level. By entering a minimum and maximum level at the bottom (the stat values for -5 levels and 30 levels respectively), the table auto-generates the band of values the program comes up with using the blending function selected from Figure 10.

|  |  |
| --- | --- |
| **Stat Level** | **Time to Kill Vanilla** |
| 1 | 20.80 |
| 2 | 7.82 |
| 3 | 4.51 |
| 4 | 3.05 |
| 5 | 2.24 |
| **6** | **1.74** |
| 7 | 1.41 |
| 8 | 1.17 |
| 9 | 1.00 |
| 10 | 0.86 |
| 11 | 0.76 |
| 12 | 0.68 |
| 13 | 0.61 |
| 14 | 0.56 |
| 15 | 0.51 |
| 16 | 0.47 |
| 17 | 0.44 |
| 18 | 0.41 |
| 19 | 0.39 |
| 20 | 0.37 |
| 21 | 0.35 |
| 22 | 0.33 |
| 23 | 0.32 |
| 24 | 0.31 |
| 25 | 0.30 |
| **26** | **0.29** |
| 27 | 0.28 |
| 28 | 0.27 |
| 29 | 0.27 |
| 30 | 0.26 |
| 31 | 0.26 |
| 32 | 0.26 |
| 33 | 0.25 |
| 34 | 0.25 |
| 35 | 0.25 |
| 36 | 0.25 |

Figure : A row from the information table that show how the stats manifest in the game. This table shows how long a player with particular damage and rate of fire levels would take to defeat a vanilla character.

The final table in the spreadsheet applies the different stat levels in a series of theoretical situations. In Figure 17 above, the table pits a leveled character up against a vanilla ship firing at point-blank range to see the best-case time it would take to kill the vanilla ship. By applying the values in this chart, data about the meaning behind those stats is available without needing to play the game and test the values, which sped up development and iteration on the stats considerably.

# Future Work

Despite all the work put into the artifact, *AllStar* is still a project, not a product. Thus, there are several areas of the game that would need to be addressed to bring the game up to shippable quality. All the game’s art assets are either from the public domain (modified or unmodified by the developer) or from another artist who had limited time to work on the project. Thus, most of the game’s art style is not cohesive, and lacks the kind of quality and beauty an indie game would need to succeed on the market today. The game’s design and balance would need a few more iterations as well. Because the project’s focus was on creating content and systems to demonstrate mastery of the programming track, less time was spent on design and balance that would need more time and talent to bring the game to market.

# Conclusion

This thesis aims to demonstrate an all-around mastery of the lessons and skills developed through the Guildhall’s software development track by creating a well-polished game prototype. By building a competitive, open-arena, two-dimensional twin-stick shooter from the ground up, polishing and optimizing the game, the artifact supports the thesis’ claim of mastery.

Completion of all tiers resulted in a complete and polished game that demonstrates a mastery of software development for games, the goal of this thesis.

# References

|  |  |
| --- | --- |
| [1] | V. Chelaru, "Rock Paper Scissors - A Method for Competitive Game Play Design," 23 January 2007. [Online]. Available: http://www.gamasutra.com/view/feature/130150/rock\_paper\_scissors\_\_a\_method\_for\_.php. [Accessed 8 September 2016]. |
| [2] | R. Terrell, "Shared-Multi-Split Screen Design," 17 June 2011. [Online]. Available: http://www.gamasutra.com/blogs/RichardTerrell/20110617/88846/SharedMultiSplit\_Screen\_Design.php. [Accessed 8 September 2016]. |
| [3] | S. Young, "Good Robot Postmortem #2: Gameplay," 19 July 2016. [Online]. Available: http://www.shamusyoung.com/twentysidedtale/?p=33343. [Accessed 8 September 2016]. |
| [4] | J. e. a. Hoffstein, "Group Report: Progression Systems," in *Project Horseshoe*, Comfort, 2014. |
| [5] | Kirby Air Ride. (GameCube). JP: HAL Laboratory, Nintendo, 2003. |
| [6] | Realm of the Mad God. (Adobe Flash). USA: Wild Shadow Studios, Deca Games, 2011. |
| [7] | Sinistar. (Arcade). USA: Williams Electronics Inc., Williams Electronics Inc., 1982. |
| [8] | Galak-Z: The Dimensional. (Microsoft Windows). JP: 17-BIT, 17-BIT, 2015. |
| [9] | Space Pirates and Zombies. (Microsoft Windows). USA: MinMax Games, MinMax Games, 2011. |

# Figures

[Figure 1: Kirby Air Ride's City Trial mode features power-ups scattered all over the level that alter the characteristics of player’s machines [5] 2](#_Toc478975003)

[Figure 2: Players battling monsters in Realm of the Mad God have to pay extremely close attention to their surroundings, as bullets come in various speeds and patterns that can end players lives instantly [6] 3](#_Toc478975004)

[Figure 3: A player avoiding Sinistar while trying to create sinibombs. The game’s open arena and obstacles match the thesis’ design [7] 3](#_Toc478975005)

[Figure 4: Galak-Z's unique handling and polish set it apart from other titles in the genre, creating the feel of actually driving a spaceship [8] 4](#_Toc478975006)

[Figure 5: A game of AllStar, featuring the two main phases of gameplay, Assembly and Challenge. 4](#_Toc478975007)

[Figure 6: The twelve power-ups, in their respective power families 5](#_Toc478975008)

[Figure 7: Concept art for the different chassis in the game. 5](#_Toc478975009)

[Figure 8: The spreadshot weapon (right) encourages players to get in as close as possible to blast enemies with as many shots as possible. The wave gun (left) outranges the spreadshot, but encourages the player to keep enemies at the focal point for optimal damage 5](#_Toc478975010)

[Figure 9: A player using their active ability to drastically increase their shot deflection. 6](#_Toc478975011)

[Figure 9: A player taking advantage of her environment to sneak up on an enemy ship. 6](#_Toc478975012)

[Figure 11: One of the minigame splash screens 6](#_Toc478975013)

[Figure 12: A UML Diagram that displays the core architecture of the program. 6](#_Toc478975014)

[Figure 13: Stage 1- The map is filled randomly with many asteroids. The GameMode then selects a random circle in the map that doesn't collide with any of the other encounters. 7](#_Toc478975015)

[Figure 14: Stage 2- The GameMode clears out any entities within the radius of the encounter, which removes any asteroids that would be colliding with this new encounter. 7](#_Toc478975016)

[Figure 15: Stage 3- The encounter is spawned in. 7](#_Toc478975017)

[Figure 16: A new encounter attempting to spawn in collides with a previous and fails. A second attempt is made that collides with no others, and succeeds. 8](#_Toc478975018)

[Figure 17: The potential stat growth curves for a stat. The green bars highlight the minimum and maximum power-up levels. 8](#_Toc478975019)

[Figure 18: A row from the stat table that demonstrates the growth of a stat's value based on the stat's level. 9](#_Toc478975020)

[Figure 19: A row from the information table that show how the stats manifest in the game. This table shows how long a player with particular damage and rate of fire levels would take to defeat a vanilla character. 10](#_Toc478975021)

1. **Anthony Cloudy** is with Southern Methodist University Guildhall, 5232 Tennyson Parkway, Building 2, Plano, Texas 75024 USA (e-mail: acloudy@smu.edu). He graduated from Southern Methodist University in 2015 (B.A. Computer Science), and has previously done contract work with Fractal Fox as a professional game developer.

   **Squirrel Eiserloh** is a game programming faculty Lecturer at SMU Guildhall, Southern Methodist University’s game development graduate program. Since he graduated from Taylor University in 1996 (B.A. Physics) he has been working as a professional game developer in the Dallas area, contributing to over a dozen commercial game titles. He co-chairs the Dallas chapter of the IGDA, and coordinates the Math for Game Programmers sessions at the annual Game Developers Conference in San Francisco. (e-mail: beiserloh@mail.smu.edu) [↑](#footnote-ref-1)