All Hands On Deck: Applying Theoretical and Practical Design to Create a Novel Game Loop

Nathan Roe
Johns Hopkins University
Engineering for Professionals, Computer Science
Baltimore, USA
nroe1@jh.edu

***Abstract*—This paper will provide a high-level overview of Video Game Design and it overlaps with Game Theory and Decision Theory. It then analyzes combining features from Roguelike and Party Games to create a new game loop structure to create a new game, *All Hands On Deck*, following the game from initial conceptualization and design through an intial prototype with play testing.**

***Index Terms*—Video Games, Simulations, Game Design, Game Theory, Party Games, Cooperative Games, Roguelike Games**

I. Introduction

Traditionally, Game Theory is the study of strategy and decision making in competitive scenarios, with the goal of mathematically modeling the decision making processes to determine opportune moves. In this way, Game Theory can be thought of as "the science of strategy" [1]. In general, the analyzed models are zero-sum competitions, where various choices are available that will lead to possible win/loss/draw scenarios for each player [2]. Using this decision tree analysis, modeling can be developed to demonstrate likeliness of winning a competitive scenario, and how each decision affects that probability. In the context of video games, there are some games that this could directly be applied to; for example, competitive *Pokémon* battle tournaments draw steep competition across the world and meet most of the classic game theory requirements.

Analyzing Video Game theory requires an expansion of this subject. While there is a broad overlap of subject matter, like defining rules and agent decision trees or looking at strategies for competition, the specifics are generally more complex as the competition may include more than two players, or one player against the game's *Artificial Intelligence* (AI). Additionally, Decision Theory applies to both traditional Game Theory and its application to Video Games. Decision Theory evaluates the set of prospects (available options) and analyzes preferences among those options for an agent in the scenario [3]. Decision Theory can be applied to evaluate which selections are more opportune in a competitive scenario, or consider how Players may interact with the set of rules within the Video Game being developed.

Practical approaches to games involve the actual programming and creation of games, like adding code to generate game mechanics for the Player or Agent to interact with. Additional practical approaches include visuals for the game, and creation of characters, models, and environments for the game. To have a complete game, it is important to consider games in both theoretical and practical ways; to fully immerse the player, the world has to be engaging and entertaining, but it is important to have challenges for the player to overcome to create additional satisfaction. Applying Game Theory allows designers to analyze mechanics for balance, making sure that challenges are appropriate for the current player skill level, or that all options for things such as in-game loadout have appropriate positives and negatives to encourage players to engage in different but viable strategies for success in games.

What sets Video Games apart from other Physics-based computer systems, such as Simulations, is the idea that games are intended to bring some sort of satisfaction to Players in order to draw engagement, using appealing visuals and creating games systems that are enjoyable and challenging to the Player. Categorizing games by genre allows for users to assess and consider games based on similarity between similarity in rules, subject matter, and even difficulty so that Players can learn from experiences and apply these lessons to improving in skills required for certain games. For example, aim and reaction time may be critical to games in the First-Person Shooter genre, while Puzzle games may require critical or non-linear thinking to come up with solutions in the context of that game's world.

One such example of a genre is the Party Game. This genre is targeted at being played with groups of users, generally together in the same room, with relatively simple mechanics and targeted at playing multiple rounds of *Cooperative* (Co-Op) or cooperative levels. Generally, these games are intended to be continually replayable and quick for Players to learn the basics [4]. The specific sub-genre this paper will look at is the Co-Op Party Game; a good example of this type of game is *Overcooked*, in which multiple players work together accomplishing tasks in a time crunch in order to prepare orders in a kitchen. The mechanics are relatively simple, but require Player coordination to succeed in the fast-paced environment. While the Players are focused on achieving their tasks, *Overcooked* throws additional environmental challenges into the mix: making players avoid or extinguish fires, or rearranging the kitchen layouts to force Players to switch up their rhythms.

This paper will evaluate combining the Co-Op Party Game

TABLE I
SUMMARY OF LEADING GAME DEVELOPMENT ENGINES

Engine	Manufacturer	Language	Platforms	Tools	Dev Friendly
Unity	Unity Technologies	C#	II-A	II-B	Yes
Unreal	Epic Games	C++	II-A	II-B	Yes
Godot	Godot Foundation (Open Source)	GDScript C#	II-A	II-B	Yes

structure with another genre, *Roguelikes*, to create a new, local Co-Op experience. The shared mechanic of Roguelike games is a repetitive gameplay loop, where Players constantly restart from the same location, but progress through the game by gaining knowledge, skills, and power-ups acquired through previous loops. These games can generally be very difficult, as they are intended to require players to iterate to progress, so players are intended to lose repeatedly. However, this repetitive nature is similar to the structure of Party Games. This game structure is intended to generate Player engagement by combining the simple mechanics requiring Player cooperation from Party Games with the skill acquisition through repetition structure of Roguelikes.

II. GAME ENGINES

Development of modern games is made possible by game engines, or development platforms with specific features made for assisting in creating games similar to current popular games. Table I contains a summary of three leading game development engines, and some of the features specific to each.

A. Supported Platforms

Each of the game engines above support multiple target platforms, with Unity and Unreal Engine including just about every mainstream gaming device. The joint platforms between each of these engines include iOS, Android, Windows, Linux, MacOS, and HTML5. Unity and Unreal Engine both contain native support for compiling on leading game devices such as Microsoft Xbox, Sony Playstation, and Nintendo Switch. External tools allow for Godot engine games to be compiled for these platforms as well, but support is not built into the base package. Both Unity and Unreal both additionally include support for compiling for Virtual Reality systems, such as the Oculus Rift.

B. Tools

Each of the game engines share a fairly similar set of tools, all aimed at providing developers the features they need to develop games from scratch. All engines compared here contain 3D object support and game test playback for game testing. Unity and Godot additionally contain separate 2D engines for simplifying the process for games that do not need depth. Unity and Unreal both additionally have their own asset marketplaces, containing both paid and free assets to assist developers in the process.

C. Review

All three engines are widely used by current developers, and provide a lot of tools and great physics engines for games. Godot has not seen widespread adoption for today's popular games, but it is user-friendly, and open source, allowing for developers to contribute to improving the engine with features they would like to see. Unity has been fairly popular, with some well-known games coming from the engine, such as *Rust* and *Kerbal Space Program*. However, uptake of Unity for professional development took a downturn in 2023 when they announce pricing changes that charge per download, which has major impacts to small developers where overall game price is lower. Because of this, Unity is seen as less friendly to indie-game developers, especially when compared to Unreal. Unreal Engine gets the highest marks here due to its widespread popularity both for indie development, as well as development for some of the most popular AAA Games, such as *Fortnite*. Both Unity and Unreal are generally considered very developer friendly, and while Unity has the largest individual developer base, Unreal has seen more uptake by existing game studios.

III. CURRENT STATE OF THE ART

A few trends are driving Video Game development across the industry. First, opportunities to monetize Video Gaming both for developers and players are driving the development of certain types of games [5]. On the Player side, monetization is led primarily by E-Sports and streaming. Because of the recent spike in viewership for both E-Sport competition as well as streaming, many large AAA game developers are working on online, multiplayer, competitive games, to try and get a part of the huge player base and viewership that games like *Fortnite* have succeeded in achieving. On the developer side, these types of games are set up well to provide microtransactions, or in-game purchases, for players who are willing to spend additional money on top of game purchases to have unique characterization in massive online communities [6].

Other recent trends include the growth of massive, open world games, where the in-game map buffers naturally to give the Player the feeling that there are endless opportunities for exploration and adventure [7]. Growth in this area is fed by the critical success of games like *Skyrim* or *The Legend of Zelda, Breath of the Wild*. These types of games have been made possible by huge advances in game engine technology, with engines like Unity and Unreal making it possible to track thousands of assets, quickly load new scenes, and optimize game performance to allow for much larger games than were previously possible.

The game concept proposed here does not follow these trends, but combines a few other popular game styles, Roguelike and Party Games. Roguelike games are increasing in popularity, with a few large success stories in recent years. Two recent examples are *Hollow Knight* and *Hades*, which both won multiple awards in their release year and achieved large

player bases. Additionally, Party Games have been popular for many years; for example, *Mario Kart 8* is the sixth-best selling game of all time, with over 80 million copies sold. Combining these genres with local-multiplayer, which is not available in many modern multiplayer games after the growth of internet-enabled games, positions this concept to be unique in the market.

IV. Game Design, Prototype, and Testing

A. Design and Storyboard

An initial storyboard of the first level, as well as the main concept for the game, is shown in Figure 1. The game centers around one loop, in which the Player, or Players, attempt to escape an island that they are told is inescapeable. In each loop, each Player must complete tasks in order to keep the boat moving towards a distant island, such as maneuvering the sail to increase speed in changing winds, or repairing damage from cannonballs or collisions with rocks. Progress within the level is shown to the Players, and the "score" is dependent on how long the ship stays afloat. The level ends when the Players either arrive at the island or sink. Levels are intended to be repeatable, with hazards spawning randomly to increase replayability, allowing Players to play in a Party Game style. Points earned in each run can be used to unlock new upgrade to survive longer or in harder levels, as well as non-functional upgrades such as new hats.

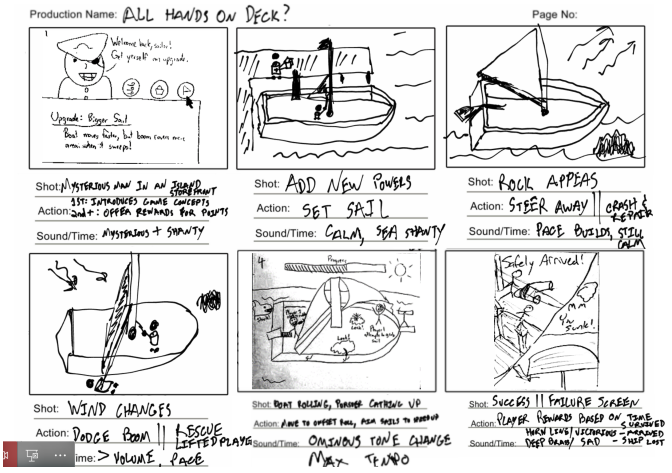

Fig. 1. Storyboard of Level 1

B. Prototyping

In order to test the concept of the main game loop, a prototype was developed for play-testing, shown in Figure 2. Several of the original features from the initial storyboard concept were implemented, including randomly spawning rock and cannon hazards, a fully sailable ship with maneuverable sail and rudder, *User Interface* (UI), and a game loop featuring initial story beats with updating interactions based on whether the Players sunk or completed the level.

Fig. 2. Prototype of Level 1

C. Play Testing

With the initial prototyping complete, initial play tests were conducted. To verify the viability of the game loop, the test plan includes tracking two metrics: play time, and discrete Player feedback. For play time, the test will track whether the game testers play for more than 4 minutes, which corresponds to the playtime required to complete the initial test level. For discrete feedback, developers will ask Players for any specific feedback on the game, and track whether this feedback aligns with the plan for game development. If feedback is in a alignment with future development plans, this is a good indication that Player concerns are with prototype work and not issues game design choices.

V. Playtest Results and Conclusion

A. Playtest Results

In testing, two Players were each given the opportunity to test the game first solo and then with one Developer. In both cases, each player wanted to play the level multiple times in multiplayer, each playing a few short playthroughs of the initial test level, until they had completed the level. This put play time for each player in the 6 to 10 minute range; a good indication of immersion for the short amount of content available.

In both playtests, the Player provided similar feedback. The first comment from each Player was that the controls were not intuitive. While there were indicators for each object in the scene that was interactable, no in-game feedback was available to tell Players which buttons they should press, and whether buttons needed to be held or a joystick moved. With some instruction, each Player was able to pick play with a few minor errors. The control scheme is built to be easily modified, so with some testing, a more intuitive control set can be implemented. Additionally, future plans will include updates to the UI to indicate interaction buttons and type (press or hold).

The other main feedback received was that the game was much better in multiplayer, and that to continue playing, more content and events need to be added. The first part of this feedback indicates the game requires additional balancing

for single Player modes, such as event spawning based on Player count. However, the general feedback was that the game was generally more exciting when there was multiple events happening at once, so it may be beneficial to focus on balancing and marketing the game as intended for multiplayer, allowing for a more chaotic atmosphere that resonated with Players. The second part of this feedback is a good sign for future development, with additional content concepts described in Section V-B.

B. Conclusion

In general, the initial game concept laid out in this paper appears to be viable. Prototyping led to a balanced game for completing an initial level in 3 to 4 attempts for pairs of Players, and kept Players engaged for that time, especially in instances of highly chaotic random draws even when those runs led to sinking. With additional playtesting and balancing, it should be possible to reach a point where 2 or more players complete initial runs in high paced runs. With the addition of ship upgrades, it should then be possible to add additional, more difficult levels, creating an engaging game loop where players do a few runs, upgrade, and do more runs in a more challenging environment.

The first planned development additions are a second, more challenging level and a few availible ship upgrades, such as an additional sail or more durable hull, that would be required for most players to complete level 2. This should prove out the full concept of the game loop, testing whether Players want to return to runs in order to gain points required to complete harder levels. The second set of additions planned include new randomly spawned events, to allow for more variability run-to-run. Events in test and conceptualization include giant shark and squid attacks, lightening strikes, ship rolling if players do not act to balance the ship, aquiring items from the sea with a net, and flyaway sails that need to be reattached. These features are in allignment with feedback received during play testing, and should give Players a more immersive experience. It therefore seems feasible that continued development could lead to a successful game achieving the initial goal of combining Party style gameplay with a Roguelike loop.

REFERENCES

[1] A. Dexit & B. Nalebuff, “Game Theory”, Econlib, [Online] Availible: https://www.econlib.org/library/Enc/GameTheory.html

[2] D. Ross, “Game Theory”, The Stanford Encyclopedia of Philosophy (Spring 2024 Edition), E. N. Zalta & U. Nodelman (eds.), [Online] Availible: https://plato.stanford.edu/entries/game-theory/.

[3] K. Steele & H. O. Sefansson, “Game Theory”, The Stanford Encyclopedia of Philosophy (Spring 2020 Edition), E. N. Zalta (eds.), [Online] Availible: https://plato.stanford.edu/entries/decision-theory/

[4] A. Mandeville, “Game Design Breakdown: Party Games”, Medium, [Online] Availible: https://alexiamandeville.medium.com/game-design-breakdown-party-games-5c2bd301cb96

[5] E. Markopoulos et. All, “Mapping the Monetiztion Challenge of Gaming in Various Domains”, HULT International Business School, [Online] Availible: https://discovery.ucl.ac.uk/id/eprint/10120905/1/Markopoulos_13-Mapping%20the%20Monetization%20Challenge%20of%20Gaming%20-Markopoulos_Evangelos_146.pdf

[6] W. Bedingfield, “Itś Not Just Loot Boxes: Predatory Monetization is Everywhere”, Wired, [Online] Availible: https://www.wired.com/story/loot-boxes-predatory-monetization-games/

[7] K. Balasubramanian, “The Evolution of Open-World Games”, Gameopedia, [Online] Availible: https://www.gameopedia.com/the-evolution-of-open-world-games/