# Introduction to Theoretical and Practical Approaches to Games

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.**

***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 structure with another genre, *Roguelikes*, to create a new, local Co-Op experience. The shared mechanic of Roguelike games

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 |

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 in to the base package. Both Unity and Unreal both additionally include support for compiling for Virtual Reality systems, such as ths 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 additonally 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 widspread addoption 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

## 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