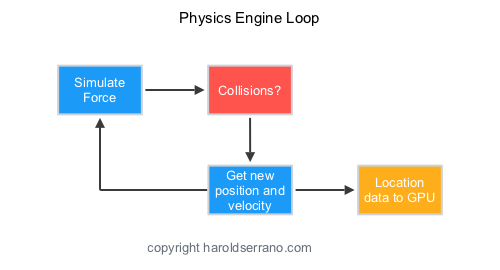
Game Design Document



# Physical Parameters:

* Physics Engine: Specify the physics engine or physics system that the game will use. Describe how it will handle aspects like gravity, collision detection, and object interactions. A physics engine is software that simulates the physical laws and behaviors of objects within a game world. It enables game developers to create realistic and dynamic simulations of how objects move, interact, and respond to forces, collisions, and other physical Phenomena.



* Core Functions: Physics engines typically handle the following core functions:
* Collision Detection: Detects when objects in the game world come into contact with each other.
* Collision Resolution: Determines how objects respond to collisions, such as bouncing, sliding, or breaking.
* Rigid Body Dynamics: Simulates the movement and rotation of solid objects like characters, vehicles, and projectiles.
* Constraints: Allows developers to impose physical constraints on objects, like joints or hinges.
* Gravity and Forces: Applies forces such as gravity, wind, and other external influences to objects in the game world.
* Friction and Material Properties: Models the friction between objects and their physical properties, like mass and density.
* Integration with Game Engines: Physics engines are often integrated into larger game engines, such as Unity3D, Unreal Engine, or Havok. Game developers can leverage these engines' physics components to create realistic and interactive environments without having to build physics simulations from scratch.
* Character Movement: Detail the physical parameters of the game's characters or entities, such as speed, acceleration, and friction. Include any special abilities or movements unique to certain characters.



* Environmental Physics: Explain how the game world will react to physics, including things like destructible environments, fluid dynamics, or dynamic weather systems.



* Ragdoll Physics: If applicable, describe how character ragdoll physics will work when characters are killed or incapacitated.



# Deployment Requirements (Hardware/Software):

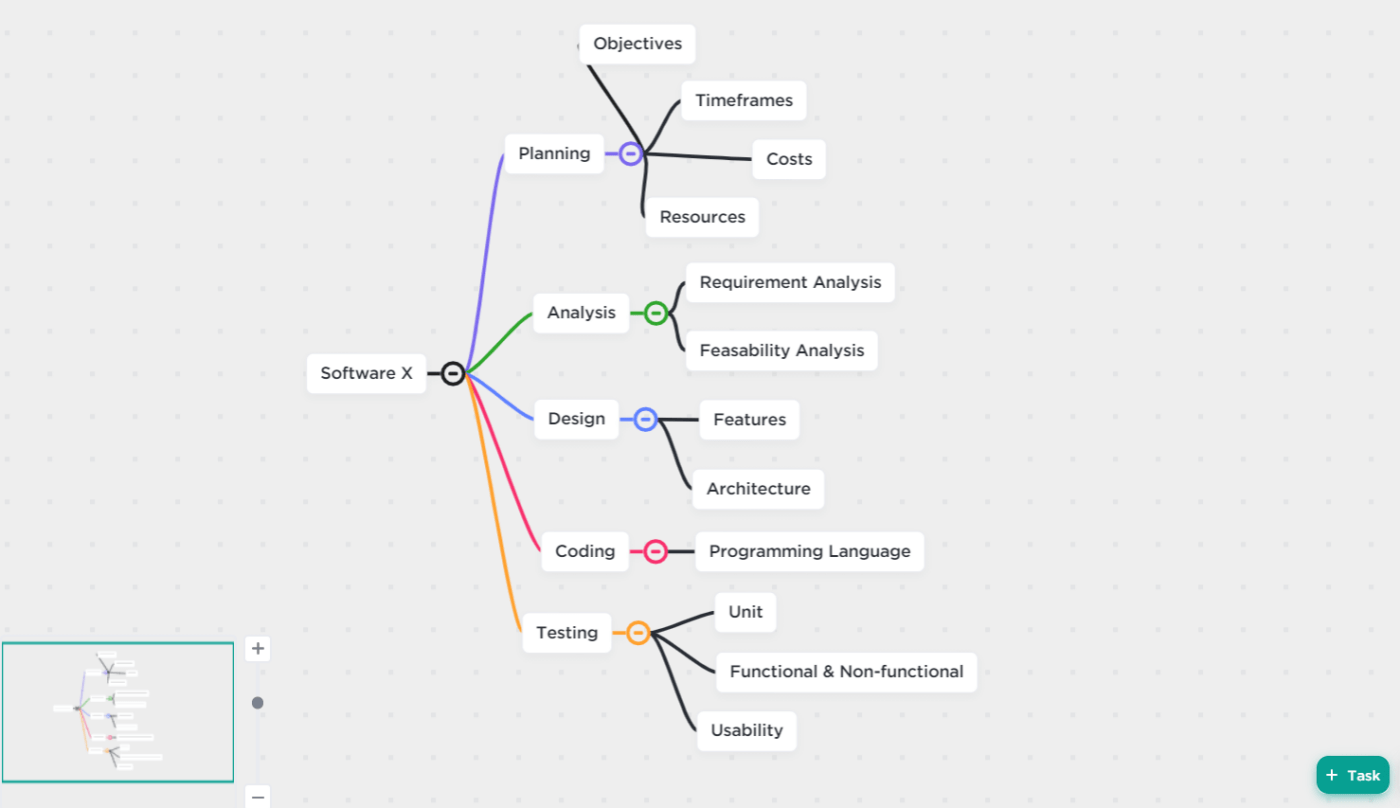
* Target Platforms: Specify the platforms the game will be released on, such as PC, consoles (e.g., PlayStation, Xbox), mobile devices (iOS, Android), or virtual reality (VR) platforms.



* System Requirements: Provide the minimum and recommended system requirements for PC or other platforms, including CPU, GPU, RAM, and storage space.



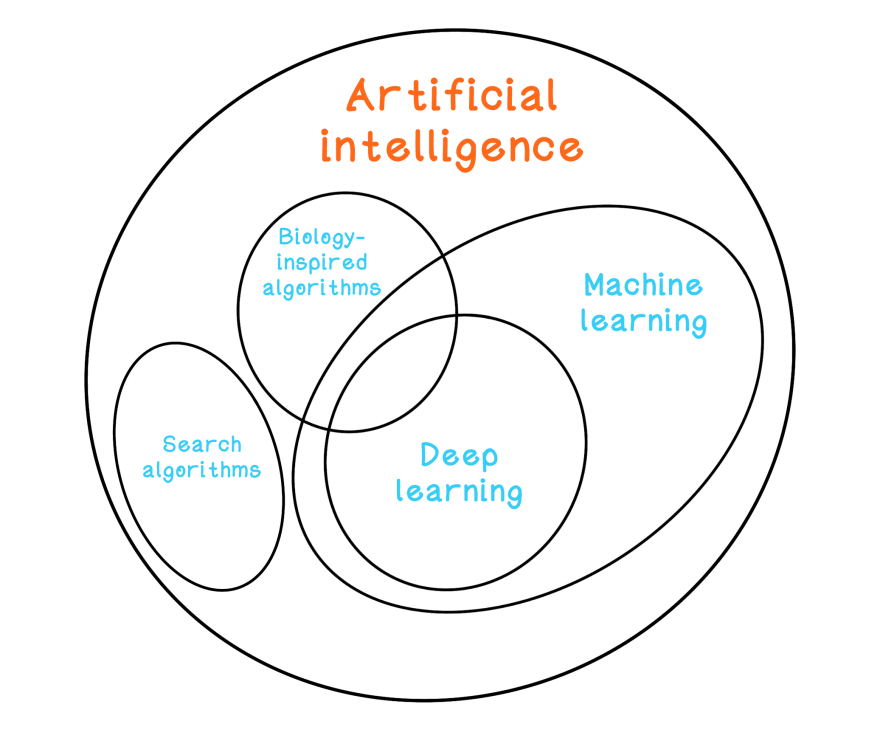
* Software Dependencies: List any third-party software or libraries the game relies on, such as game engines (e.g., Unity, Unreal Engine) and development tools.



* Network Requirements: Describe the network architecture and any server requirements for online multiplayer features.
* Localization: If the game will be available in multiple languages, outline the localization process and any software/tools needed for translation.

# Algorithm Reasoning and Scripting:

* AI Algorithms: Explain the algorithms and logic used for non-player character (NPC) or enemy behavior. Discuss pathfinding, decision-making, and any learning algorithms (e.g., machine learning).



* Gameplay Mechanics: Detail the algorithms behind key gameplay features, such as combat systems, puzzle-solving, or crafting mechanics. Gameplay mechanics refer to the rules, systems, and interactions that define how a video game is played. These mechanics are the core elements that players engage with to progress, solve challenges, and enjoy the game. Gameplay mechanics are a fundamental aspect of game design and greatly influence the player's experience. Here are some common examples of gameplay mechanics and their descriptions:

## Movement and Navigation:

* Character Movement: How the player's character or avatar moves within the game world, including walking, running, jumping, crouching, and more.
* Vehicle Control: If the game involves vehicles (e.g., cars, spaceships, or mounts), the mechanics for controlling them.

## Combat and Interaction:

* Attack and Defense: The mechanics for attacking and defending, which may involve combat with weapons, spells, or other means.
* Inventory and Equipment: How the player manages items, weapons, and gear, including inventory management, equipping, and upgrading.
* Crafting and Item Usage: If the game includes crafting, potion-making, or item usage, the mechanics for these actions.
* Dialogues and Choices: If the game features branching narratives, the mechanics for making choices and interacting with non-player characters (NPCs).

## Puzzles and Challenges:

* Environmental Puzzles: Mechanics for solving puzzles in the game world, which may involve manipulating objects, finding clues, or completing challenges.
* Logic and Riddles: How players engage with logical puzzles or riddles within the game.

## Progression and Rewards:

* Experience Points and Leveling: Mechanics for character progression, including earning experience points and leveling up.
* Achievements and Collectibles: How players can earn achievements or collect in-game items or trophies.
* Quests and Objectives: Mechanics for setting and completing quests or objectives.

## Resource Management:

* Health and Energy: Mechanics for managing a character's health and energy, including health packs, food, or rest.
* Currency and Economy: How in-game currency is earned, spent, and managed.

### Physics and Simulations:

* Physics-Based Mechanics: For games with realistic physics, mechanics for object interactions, gravity, and other physical simulations.
* Simulations: How the game models aspects like weather, day-night cycles, or ecosystems.

## Multiplayer and Social Mechanics:

* Cooperative Play: Mechanics for working together with other players in co-op games.
* Competitive Play: Mechanics for player-vs-player interactions and competition.
* Social Interactions: Mechanics for player-to-player communication, trading, or alliances.

## Time Management:

* Time-Dependent Mechanics: Games that involve managing time or schedules have mechanics related to planning and execution.

### Stealth and Infiltration:

* Stealth Mechanics: For games that involve sneaking, hiding, or espionage, the mechanics for remaining undetected and outsmarting enemies.

## Platforming and Parkour:

* Platforming Mechanics: Mechanics for precise jumping and navigating platforms, common in platformer and 3D adventure games.

### Sandbox and Building:

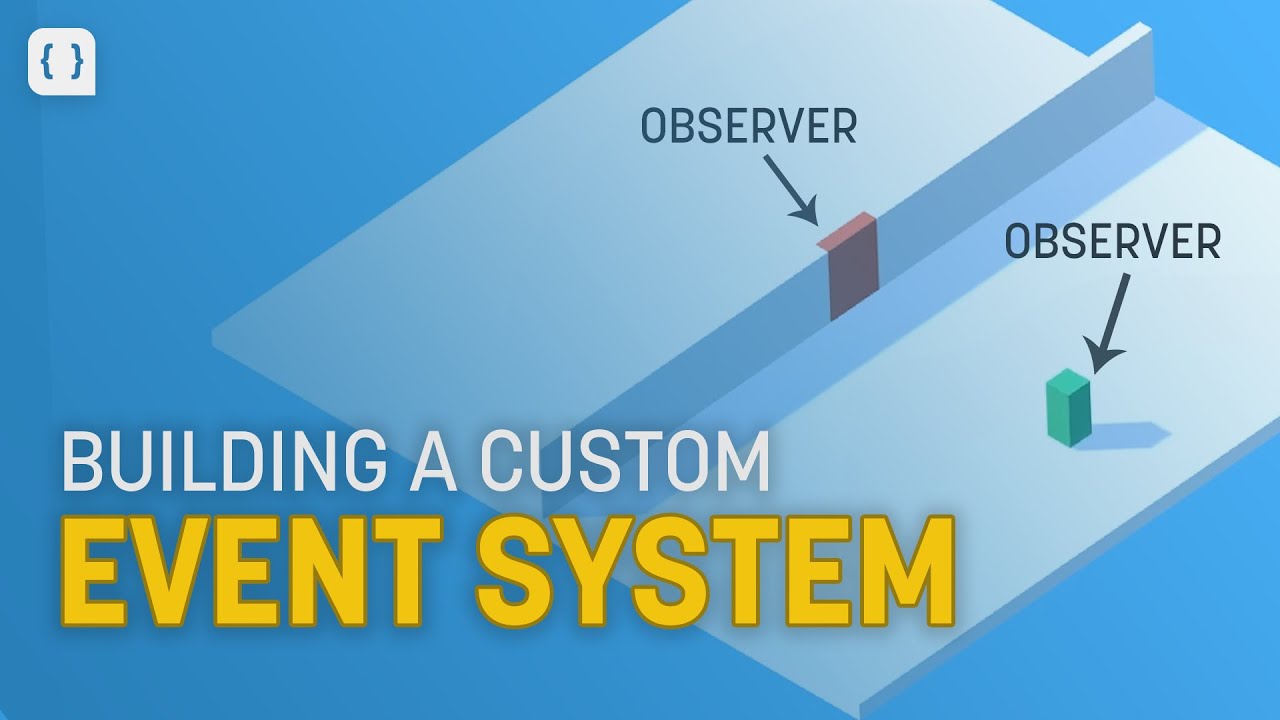
* Building Mechanics: In sandbox games, mechanics for constructing structures or creating content within the game world.

Gameplay mechanics can vary widely depending on the genre and design goals of the game. Effective game design involves selecting, balancing, and integrating these mechanics to create engaging and enjoyable gameplay experiences that align with the game's overall vision and objectives.

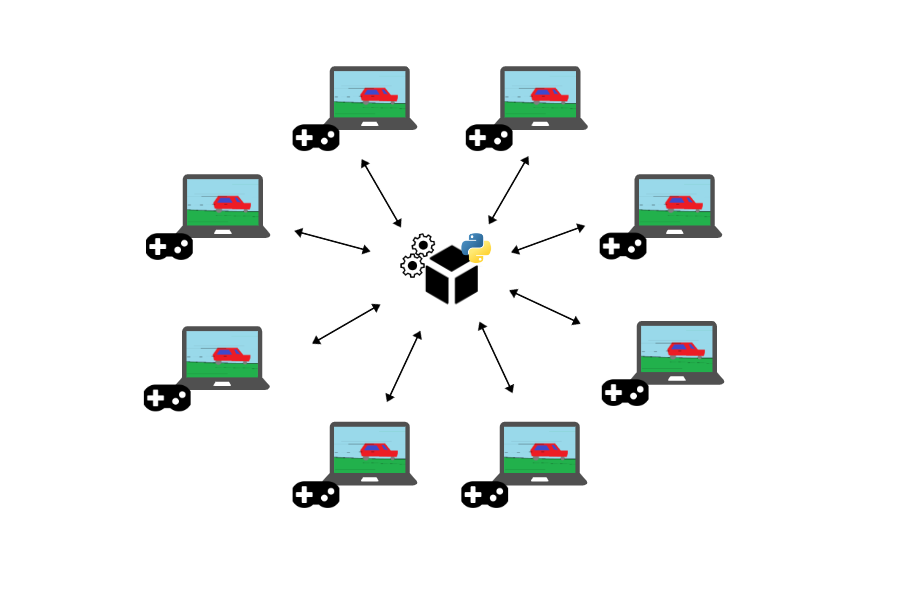
* Scripting Language: Identify the scripting language(s) used for implementing game logic and interactions (e.g., C#, Lua, or proprietary scripting languages).



* Event Systems: Describe how events and triggers are handled in the game. This may include cut scene scripting, event-driven gameplay, or dialogue systems.



* Server-Side Logic: If the game involves multiplayer, outline the server-side scripting for functions like matchmaking, game balance, and anti-cheat measures.



* Game Rules: Provide a comprehensive explanation of the rules and conditions governing the game, from scoring to win/lose conditions and progression.



By addressing these aspects in your GDD, you create a solid foundation for the game's development. The physical parameters, deployment requirements, algorithms, and scripting details are essential for guiding the technical implementation and ensuring that the game functions as intended on the chosen platforms and with the desired gameplay features. Additionally, it aids in communication between various development team members and stakeholders.