Name Adam Cobb   Mark ______________________/50

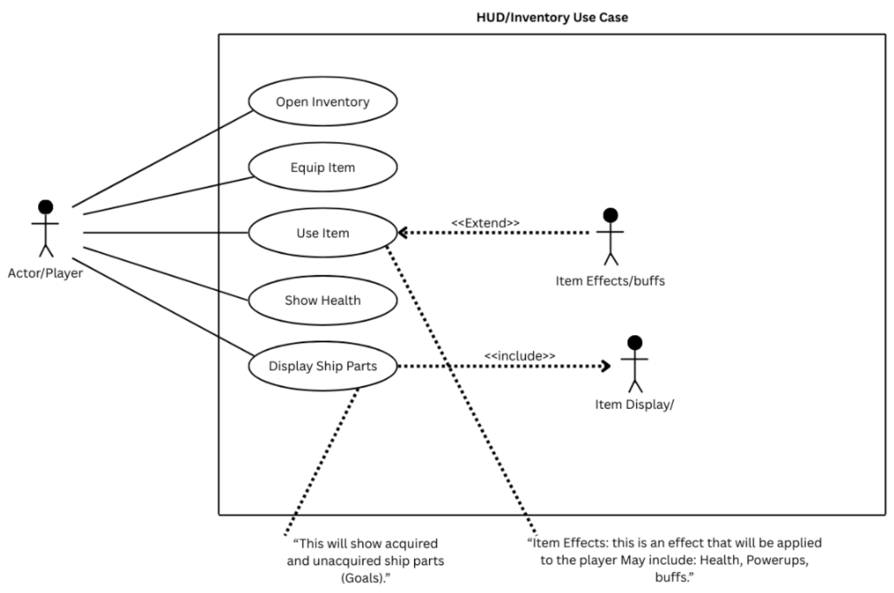
1. Brief introduction __/3

Where's My Spaceship is a 2D action-adventure game featuring multi-era exploration, platforming, and dynamic combat. As part of the development team, I am responsible for designing and implementing key game interface elements including the **Inventory, HUD Main Menu, and Pause Menu**. These features are essential for enhancing the player's experience by providing unique features for the HUD in the different respective era's.

The **Inventory system** allows players to manage collected items, era-specific artifacts, and upgrades. The **HUD** provides real-time gameplay information such as health, abilities, collected ship parts, and current objectives. The **Main Menu** offers essential navigation for starting a new game, loading progress, or adjusting game settings, while the **Pause Menu** ensures players can take breaks, access controls, or modify settings without losing progress.

These interface components are integral to creating an immersive and user-friendly experience as players guide **Dr. Tempus Rift** through prehistoric landscapes, medieval castles/fields, and cyberpunk cities in search of his lost spaceship.

2. Use case diagram with scenario __14


HUD/Inventory Use Case
Actor/Player
Open Inventory
Equip Item
Use Item
Show Health
Display Ship Parts
<<Extend>>
Item Effects/buffs
<<include>>
Item Display/
"This will show acquired and unacquired ship parts (Goals)."
"Item Effects: this is an effect that will be applied to the player May include: Health, Powerups, buffs."

Scenarios

Name: HUD/Inventory Use Case Diagram
Summary: HUD Diagram: Shows the player as an external actor interacting with the HUD system. The system contains functions that display real-time game information, such as health, collected items, abilities, objectives, and timers. The diagram emphasizes monitoring and feedback, not direct gameplay actions.

Actors:

Actor (Outside the Box):

- Player – needs real-time information during gameplay.

System (Inside the Box):

- Display Health
- Show Collected/Uncollected Ship Parts
- Show Equip Item
- Shows Inventory

Preconditions:

HUD:

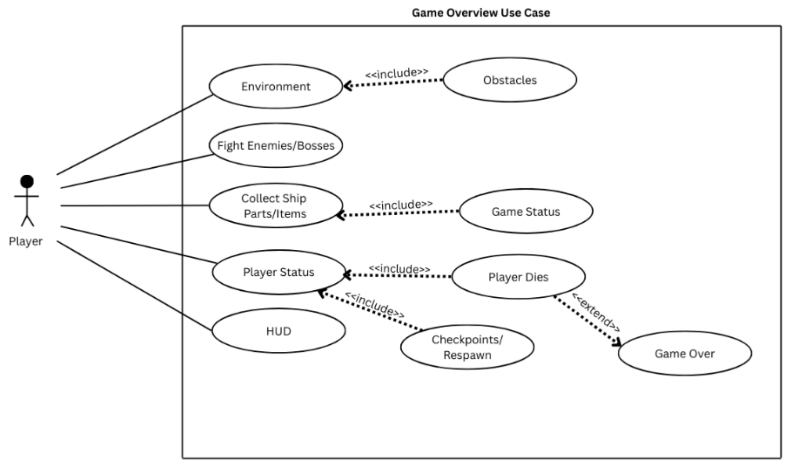
- The game is running.
- Player character exists and has initialized stats (health, abilities, inventory).
- Level or mission data is loaded.
- HUD system is initialized and ready to display information.

Basic sequence:

Step 1: Initialize and display HUD elements (health, abilities, score, timer).
Step 2: Continuously update HUD based on player actions and game events.
Step 3: Reflect changes immediately (e.g., health drops, items collected, objectives updated).
Step 4: Maintain display until game ends or HUD is turned off.Exceptions:

Post conditions:
All relevant game information (health, score, abilities, collected items, timer) is **accurately displayed** to the player.
Player has **up-to-date feedback** for decision-making during gameplay.
HUD remains active and responsive throughout gameplay until the game ends or the HUD is disabled.
Priority: 2*

ID: UC01



Name: Game Overview Use Case Diagram
Summary: Game Overview / Main Loop Diagram: Shows the player as an external actor interacting with the main game system. The system encapsulates the core gameplay mechanics, including exploration, combat, puzzle-solving, item collection, inventory updates, and status checks. It also includes sub-processes like Game Over and Victory, demonstrating the game's response to the player's state.

Actors:

Actor (Outside the Box):

- Player – explores, fights, collects items, solves puzzles.

System (Inside the Box):

- Explore Environment (← Obstacles)
- Fight Enemies / Bosses
- Collect Ship Parts / Items (← Game Status)
- Check Player Status (Alive / Dead) (← Player dies ← Game Over) and (← Checkpoints and Respawn)

Preconditions:

Game Overview / Main Game Loop:

- Game engine has started.
- Player has control of the character.

- Environment, enemies, and items are loaded.
- Necessary systems (HUD, inventory, combat, puzzles) are initialized.

Basic sequence:

- **Step 1: Load game environment, player character, enemies, items, and systems.**
- **Step 2: Accept player actions (movement, attacks, interactions).**

- **Step 3: Update game state based on player actions (combat outcomes, puzzle completion, inventory changes).**

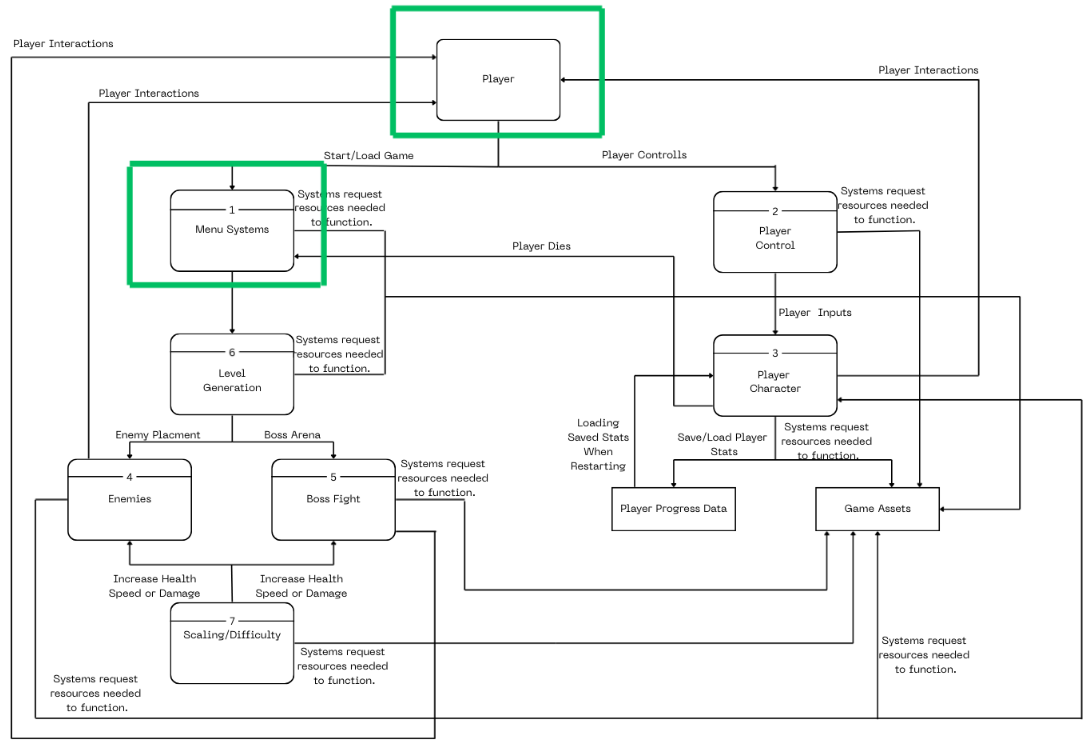
- **Step 4: Check player status (alive/dead) and game conditions (victory/mission complete).**

- **Step 5: Trigger feedback through HUD, sounds, animations, or Game Over / Victory screens.**

- **Step 6: Repeat loop until game ends or player quits. Post conditions:** Calculated value is displayed.

  Priority: 1*
  ID: UC02

3. Data Flow diagram(s) from Level 0 to process description for your feature ______14


Player
Player Interactions
Player Interactions
Player Interactions
Start/Load Game
Player Controlls
1
Menu Systems
Systems request resources needed to function.
2
Player Control
Systems request resources needed to function.
Player Dies
Player Inputs
6
Level Generation
Systems request resources needed to function.
3
Player Character
Loading Saved Stats When Restarting
Save/Load Player Stats
Systems request resources needed to function.
Enemy Placment
Boss Arena
4
Enemies
5
Boss Fight
Systems request resources needed to function.
Player Progress Data
Game Assets
Increase Health Speed or Damage
Increase Health Speed or Damage
7
Scaling/Difficulty
Systems request resources needed to function.
Systems request resources needed to function.
Systems request resources needed to function.

1. **Player Interactions**

   - The player is the primary actor who initiates interactions such as starting the game, controlling the character, and providing inputs.

2. **Menu Systems (1)**

   - Handles starting or loading a game.
   - Requests necessary resources to function (menus, UI assets).
   - Passes control to **Level Generation (6)** to prepare the game world.

3. **Player Control (2)**

   - Receives player inputs (movement, actions).
   - Requests required system resources for smooth gameplay.
   - Sends inputs to **Player Character (3)** for execution.

4. **Player Character (3)**
   - Represents the in-game avatar controlled by the player.
   - Receives inputs from **Player Control** and updates the game state.
   - Communicates with **Player Progress Data** to save/load stats.
   - Uses **Game Assets** (models, animations, sounds) as needed.
5. **Enemies (4) & Boss Fight (5)**
   - Enemies are spawned according to **Level Generation (6)**.
   - Boss Fight represents special encounters (Boss Arena).
   - Both receive adjustments from **Scaling/Difficulty (7)**, which can modify health, speed, or damage.
6. **Level Generation (6)**
   - Responsible for creating the game environment, placing enemies, and setting up boss arenas.
   - Requests system resources to function properly.
7. **Scaling / Difficulty (7)**
   - Dynamically adjusts gameplay difficulty, modifying stats for **Enemies** and **Bosses** based on progression or settings.
   - Ensures the game remains challenging but balanced.
8. **Player Progress Data**
   - Saves and loads player stats, inventory, and progress.
   - Provides information to **Player Character** when restarting or continuing the game.
9. **Game Assets**
   - Provides required resources such as models, sounds, textures, and animations to all systems that need them.
10. **Game Flow**
    - Player starts the game via **Menu Systems** → Level is generated → Enemies and bosses are placed → Player interacts via controls → Game state updates

dynamically → Progress is saved → Difficulty is scaled as needed → Feedback loops back to player via character, HUD, and gameplay results.

4. Acceptance Tests ______9

Example: HUD Feature Test

Input: Simulated gameplay session where player takes damage, collects items, gains abilities, and completes objectives.

Output (HUD display):

- **Health: Never below 0, never above maximum set value.**
- **Items/Parts Collected: Count updates correctly each time an item is picked up.**
- **Objectives/Score: Updates consistently with game progress.**

Boundary Cases:

- player health starts at full → decrements correctly to 0.
- Player collects 0 items → HUD shows empty inventory.
- Player pauses game → HUD freezes but does not reset.

Example: Game Overview / Main Loop Test

Input: Simulated playthrough with exploration, combat, puzzle solving, and boss fight.
Output (Game System behavior):

- **Environment:** Generated correctly at start; no missing assets.
- **Enemies:** Spawn in valid locations; never overlap incorrectly.
- **Boss Fight:** Triggers only once per level.
- **Game Over / Victory:** Triggers exactly once when conditions met.

Boundary Cases:

- Player starts with no saved data → new game initializes correctly.
- Player health = 0 at spawn → immediate Game Over triggered.
- Player collects all items → triggers Victory state.

HUD Feature Test Cases

Test ID	Description	Input	Expected Output	Boundary Case

HUD-01	Health display	Player takes damage	Health decreases but never below 0	Health starts full, decrements correctly to 0
HUD-02	Health max cap	Player collects healing items	Health increases but never above max	Healing at full health → stays max
HUD-03	Item collection	Player picks up item	Item count increases	Player collects 0 items → HUD shows empty inventory
HUD-04	Objective/score update	Player completes objective	Score/Objective progress updates	Completing final objective → triggers win condition
HUD-05	HUD freeze on pause	Player pauses game	HUD display freezes (does not reset)	Game paused → HUD state unchanged

Game Overview / Main Loop Test Cases

Test ID	Description	Input	Expected Output	Boundary Case
GOV-01	Environment generation	Start new game	Level generates fully, no missing assets	Player starts with no saved data → new game initializes correctly
GOV-02	Enemy spawning	Enter combat area	Enemies spawn in valid locations, no overlap	Edge of map spawning → still valid placement
GOV-03	Boss fight trigger	Player reaches boss room	Boss spawns and triggers fight once	Boss triggers only once per level
GOV-04	Game over condition	Player health reaches 0	Game Over screen shown exactly once	Player health = 0 at spawn → immediate Game Over
GOV-05	Victory condition	Player collects all items	Victory state triggered exactly once	Collecting all items triggers win sequence

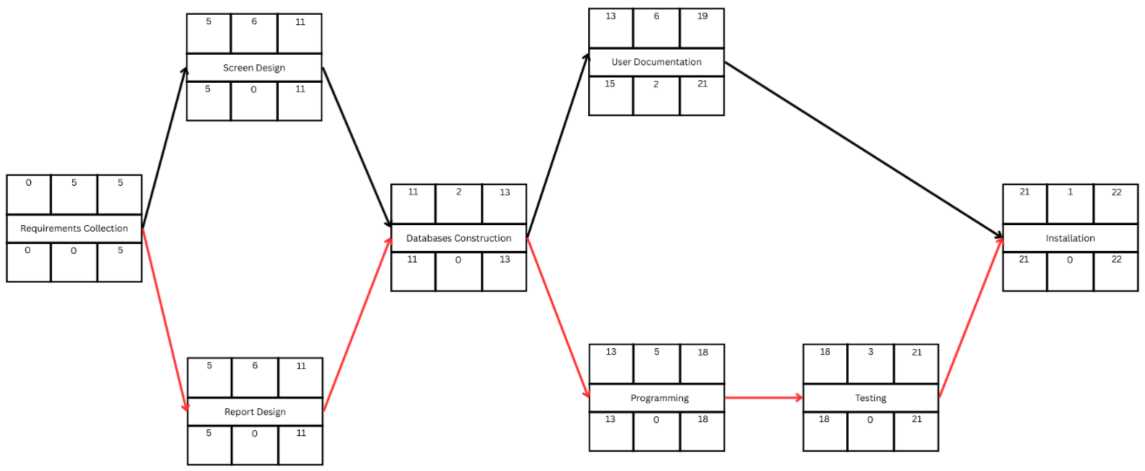
5. Timeline ________/10

Work items

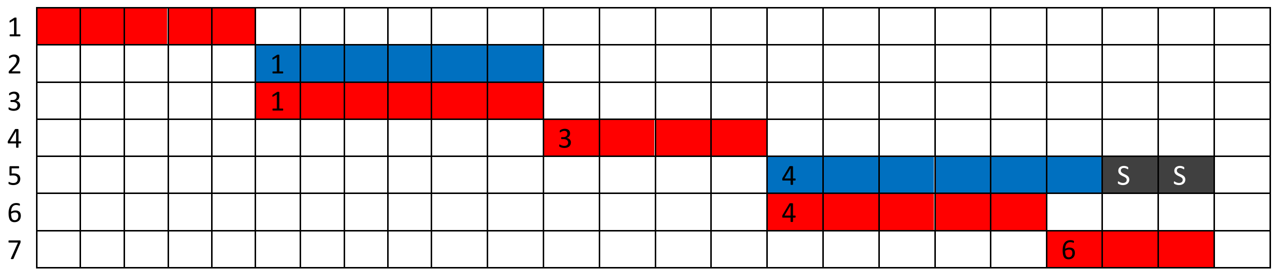
Task	Duration (PWks)	Predecessor Task(s)	Early Start	Early Finish	Late Start	Slack Time
1. Requirements Collection	5	-	0	5	0	0

2. Screen Design	6	1	5	11	5	0
3. Report Design	6	1	5	11	5	0
4. Database Construction	2	2,3	11	13	11	0
5. User Documentation	6	4	13	19	15	2
6. Programming	5	4	18	21	18	0
7. Testing	3	6	18	21	18	0
8. Installation	1	5, 7	21	22	21	0

Pert diagram



Gantt timeline



8																								7
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24