Prototype Report

Guillaume St-Pierre – 101066038

Description

In this game, you are a diver trying to find the great underwater treasure buried deep into the ruins. With your limited set of tools, you must dive deeper and deeper into the ruins, collect treasure, fend of monsters and buy new tools to hopefully find the great treasure at the end of the ruins.

The prototype implements most of the logic for the game, but some of the logic needs to be tweaked so it feels great to play and the bugs need to be ironed out.

Here are the controls for this current prototype:

- Moving is done with the A and D keys. As you are an underwater diver, water will apply resistance to your movements, and you will slow down if you don't hold the keys.
- You can move up in the water using Spacebar. Gravity is still applied when moving up and you will slowly sink back down if not holding it.
- You can move your current weapon's aim using the mouse and attack with a mouse click.
- You can select your weapon with the 1, 2 or 3 key. The Harpoon is weapon 1, the pistol is weapon 2 and the laser is weapon 3. Weapon 2 and 3 are locked at the start of the game.
    - CTRL + the weapon key will unlock the given weapon.
    - SHIFT + the weapon key will trigger an update on that weapon, increasing its stats.

To play the game, try moving around and attacking the monsters. You may collect treasure by moving next to them and leave the level by moving to the arch-like structure on the right side of the level. The game has two identical levels that loop, you can test that treasure does not respawn between levels by going through the loop.

The player currently cannot attack when being hit. This is a limitation of the state machine on the player and will be fixed for the final game.

Collision detection only partially works right now, as we have yet to see how to properly implement it. The TA in my lab session suggested I wait for a correct implementation and keep my partially broken one.

If you use the debugger, you will likely teleport outside the level due to the gravity effect. This is due to how the deltatime is use in the euler integration and should not affect gameplay.

Requirements

Technical requirements.

1. Written in C++ using OpenGL to render; readable code with no fatal bugs.

    a. The game is indeed written in C++, most of the code is document with the exception of some of the early member variables and methods. The code is properly formatted and I hit no major bugs when testing the game.

2. All movement handled through transformations.
    a. This is indeed the case as all movement is handled using the transformations in the render methods of each entity. A lot of inheritance is used to render the entities and transformation are used throughout.
3. At least one use of physically-based motion (with gravity and momentum-based collisions).
    a. The player, treasures and powerups are all affected by gravity through their parent object. The player can fight against gravity through the controls or by using their ability to stick to walls.
4. Collision detection between game entities and the walls and platforms of the level.
    a. While collision detection is far from perfect, it does nominally work. The player will stop when trying to move against walls, the floor or the ceiling. With the issues in collision detection, it is possible to move inside the walls, this will be fixed soon.
5. At least one instance of hierarchical transformations for a compound object (e.g., a cannon with swiveling barrel).
    a. The player has weapons which are hierarchical transformations. The player can both move around and flip and the weapons will stay attached to the arm of the player sprite. Attacking also triggers the weapon's attack animation using these transforms.
6. A gameworld larger than the screen, with ability to scroll around and see different parts.
    a. The camera is centered on the player at all time. When moving, the world will be moved accordingly so the player is always centered. The current level does not demonstrate that extremely well, but it does work.
7. At least one enemy that navigates using path planning.
    a. This is not yet implemented. I intend to use path planning as my line of sight mechanic. Monsters which are able to attack the player will check if they can see them using path planning. Furthermore, the jellyfish (The big fish sprite) will use path planning as well, moving to the player's node and then using the chase steering behavior when they get there.
8. At least one game entity that moves using chase, pursuit, flee, or escape behaviours.
    a. The small fish currently charges the player using chase. The Jellyfish will attempt to flee when the player gets too close, but this has not yet been implemented.
9. Finite state machine controlling overall game (e.g., loading screen, pause, break from action between levels).
    a. The state machine exists and currently powers the game transition. Pausing will be done using that state as well.
10. At least two instances of particle systems used.
    a. We have yet to learn about particle systems, so those have not yet been implemented. They will be used to generate the smoker's smoke and air

escaping the player to show we are indeed underwater (With more air leaving when hit)

Gameplay requirements.

1. At least three different enemy types with distinct behaviours.
    - We only have two enemies implemented right now, but a third one will be included soon. The fish charges the player when they are in their line of sight. The Jellyfish will hurl electricity at the player. The smoker will create a wall of smoke that blocks player progression.
2. At least three levels with different layouts.
    - Optionally, different finishing conditions for each level: e.g., clear enemies, destroy key enemy, collect all keys, collect key and reach exit. You can probably think of many others.
        - We currently have two identical levels which are completed by getting to the exit. Levels are also exited with the player has not enough air to continue. Though the logic exists, this is not yet implemented.
3. At least two distinct weapons the player can use.
    - We have three weapons. The harpoon does good damage but is only used in close quarters. The pistol has low range and low damage but shoots fast. The laser has a huge range and size and does a massive amount of damage but is slow to recharge.
4. Use of currency for player to buy upgrades. (Or, some clever way to otherwise get the player to make long-term decisions.)
    - Upgrades will be bought using treasure, but this has yet to be implemented. Upgrades exists, but they cost nothing.
    - 4a. optionally, RPG-style upgrade paths for the player: additional weapons, shields, speed boosters, and more.
        - The player can unlock two new weapons throughout the game and both weapons are upgraded individually. This is implemented, though the HUD for it does not exist yet.
5. A HUD showing useful information: current cash, health, other game elements of interest.
    - The HUD has yet to be created. I am waiting on the class that will teach us how to draw text on screen properly.
6. At least one interesting powerup with a timer.
    - Powerups are not implemented right now, though the logic for it does exist. It will be a matter of creating the entity and making it pickupable.
7. Challenges that reward planning as well as reflexes.
    - Only reflex based challenges exist right now in the form of the fish charging the player. More planning based challenges will come through level design when we design better levels.

Planned features

Water physics-based movement	COMPLETED
Camera controls	COMPLETED
Collision detection on walls	IN PROGRESS
Can stick to walls and floor	COMPLETED
HUD	NOT STARTED
State machine for levels and menus	COMPLETED
Menu for starting the game	NOT STARTED
Pause menu	NOT STARTED
Player loses air as time goes one	COMPLETED
Player loses air and treasure when hit	COMPLETED
Harpoon weapon	COMPLETED
Pistol weapon	COMPLETED
Laser weapon	COMPLETED
Weapons are represented on the player	COMPLETED
Collectable treasure that do not respawn	COMPLETED
Unlockable weapons	COMPLETED
Upgradable weapons	COMPLETED
Upgrade and unlock menu	NOT STARTED
Fish Enemy that charges at the player	IN PROGRESS
Smoker enemy that create a wall of smoke	NOT STARTED
Jellyfish enemy that shoots electricity	IN PROGRESS
Enemies can drop treasure or powerups	COMPLETED
Powerups that can be picked up	IN PROGRESS
Level can be exited	COMPLETED

Schedule

Week	Plans
1st week – 1st half	Implement all menus so the game is now in a less tech-demo state and more in a playable state. Attempt to write things on the screen.
1st week – 2nd half	Create all the transitions for the various states of the game. The game should be startable, pausable and closable. Pressing escape mid-level should open the pause menu and resume the level if the menu is closed.
2nd week – 1st half	Create the path planning behavior for the fish entity. It should allow the fish to “see” the player, but not through walls. Create a generic djsiktra algorithm system that can be reused.
2nd week – 2nd half	Create the Jellyfish AI. It should flee when it sees the player through the fish line of sight system from the first half and shoot

	electricity if the player if far enough. It should flee in a “stupid” way, getting stuck in walls is intended.
3rd week – 1st half	Add levels and make them playable. At this point, we should have everything we need features wise and can focus on making good levels. The smoker enemy should also be added around this time, depending on when we see particle systems in class. This enemy is pretty dumb and does not move, so it won’t be too difficult to create.
3rd week – 2nd half	Polish and playtest the levels. Implement the air particle system if time permits.
4th week	Final polish, make the best out of the remaining time.