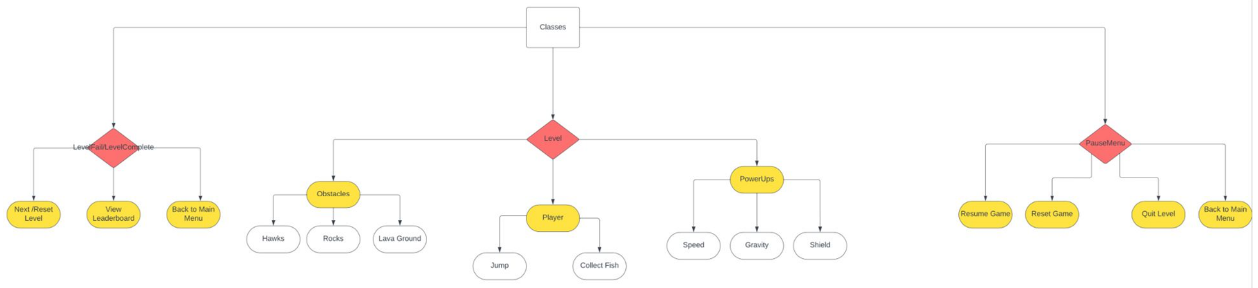
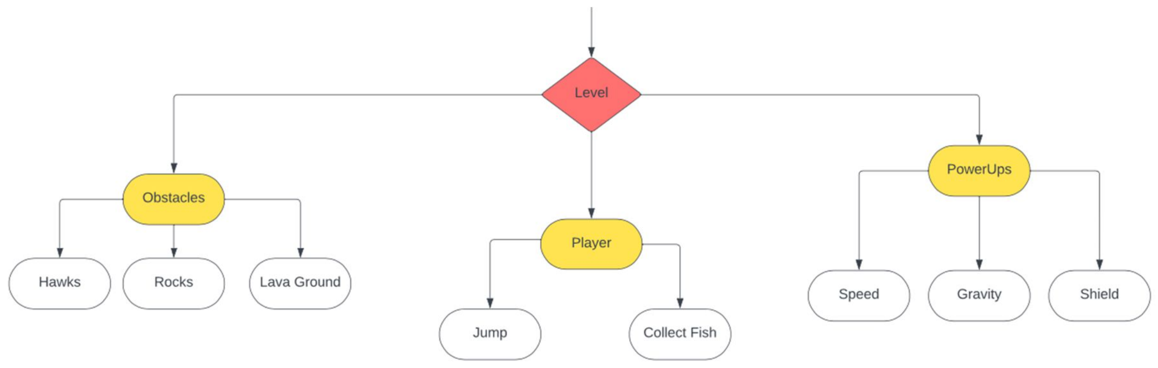
MEMO Number: TechnicalDocument-CMSC447-Himalayans
DATE: March 24, 2024
TO: Nick Allgood
FROM: CMSC447-04 Team Himalayans <everest-the-olympicat@googlegroups.com>
Anna Pham, Connor Gilger, David Middour, Jess Turner, Samuel Oyeneyin
SUBJECT: Technical Document

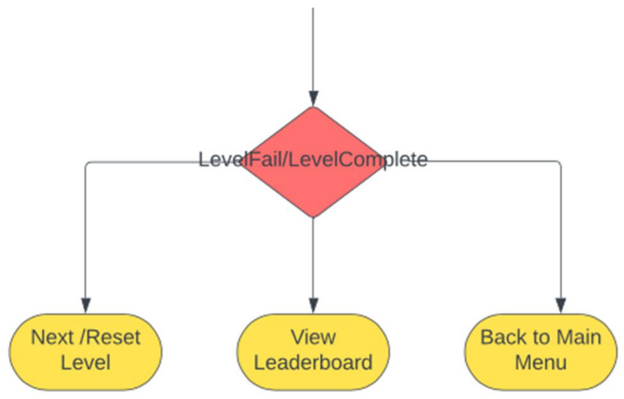
CLASS DIAGRAM



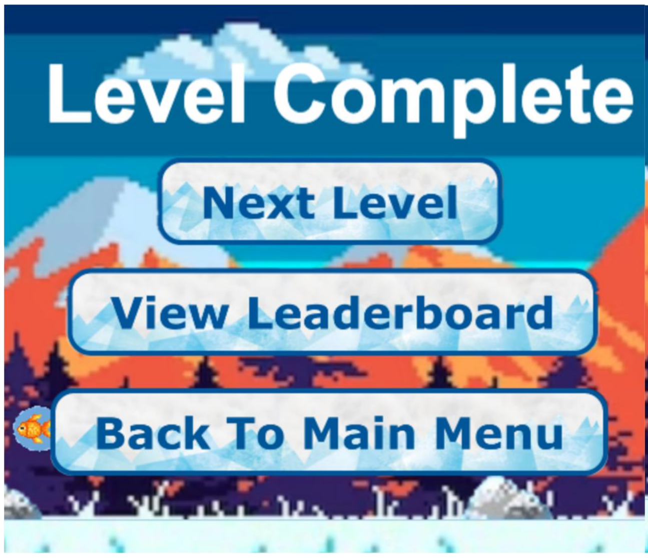
The above diagram shows the layout for each level. Each level contains a class for the level itself, a class for the pause menu, a class for the level complete screen, and a class for the level fail screen. The level complete screen and level fail screen are combined in the diagram since they are exactly the same except the player is allowed to proceed to the next level if they complete the level or the player is allowed to reset the level if they fail it. The third level complete screen is exactly the same as the level fail screen since there is no next level to go to.

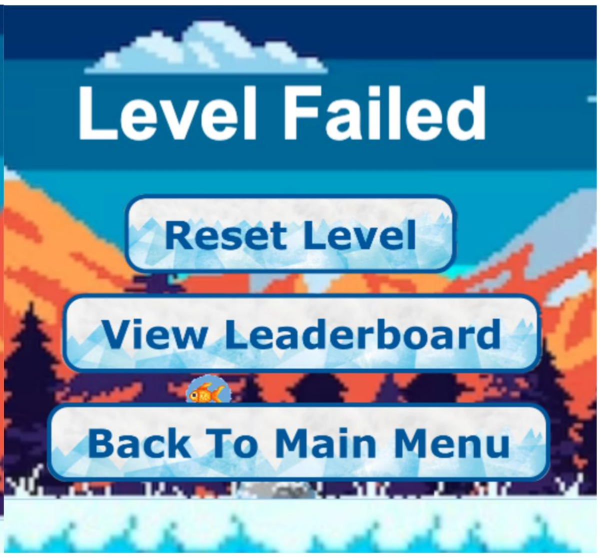


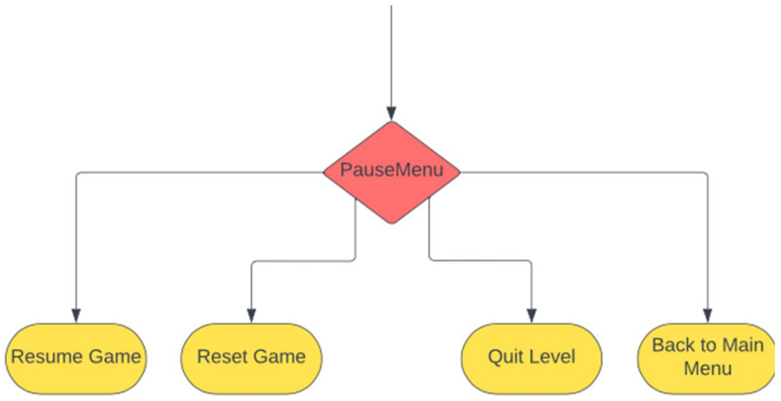
Looking at the level class, there are three categories: player, obstacles, and power ups. Within the player category, the player is allowed to jump and collect fish for points. In the obstacles category, there are multiple different types of obstacles, such as rocks, birds, etc., that will spawn depending on the level the player is on. All three levels have the same power ups that affect speed, gravity, and invincibility. The player can pick up a jump boost to jump higher, a speed boost to move faster, or a shield that will protect the player from obstacles.



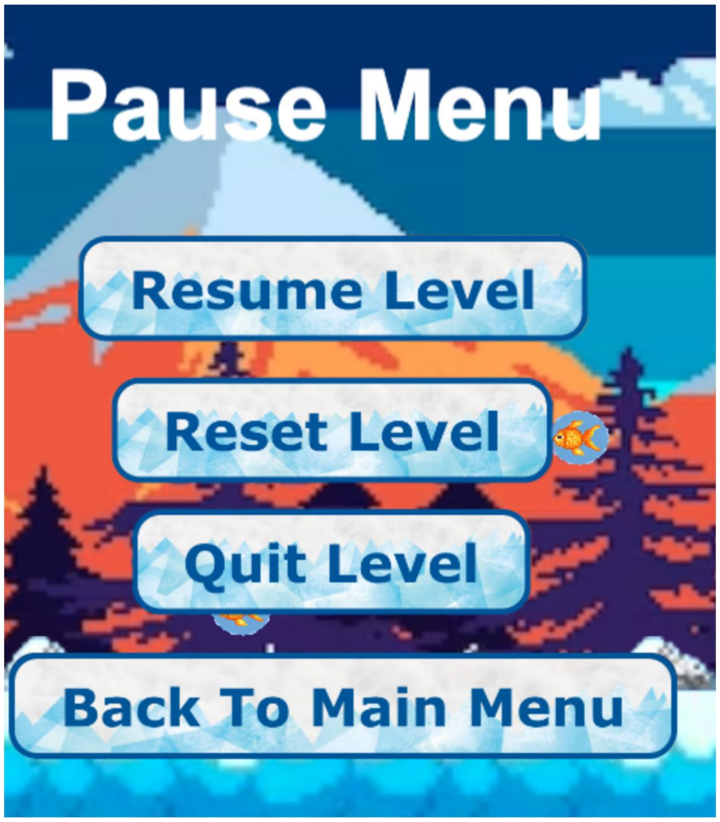
The level fail and level complete classes contain three buttons that the player can navigate to reset the level they are currently on or go to the next level, view the leaderboard, or go back to the main menu. The below images are what the complete and failed screens look like.







The pause menu class contains four buttons that allow the player to resume the level they are currently on, reset the level, quit the level to go back to the level options, or go back to the main menu. The below image is what the pause menu looks like.



DATABASE

Jessica TurnerSamuel Oyeneyin
(Jess todo: update the Database diagram with new column names)

API CALLS

For an easier to read version, I recommend looking at the `Backend/README` on our repository: https://github.com/David-B-M/cmsc447-sp2024-himalayans/blob/main/Backend/README.md

✓ Add/Create user

-

```
POST add_user
parameter: username
```

headers: None required

Returns: json
Example:
```
  {
     "user_id": 1,
     "msg": "Successfully added user `jess` to database! :D"
  }
```

- •Usage: Frontend StartGame page.
- example call:

•POST add_user?username=test1

- •when user presses [New Game] and enters their name, send it to the backend to validate and save!

✅ Load (read all) users

•
GET /load_users

- •Usage: Frontend StartGame page.
    - return the information from the users table so that the frontend can display "save files" (usernames and the levels they reached + maybe their highscore).
- Response Elements:
    - ok: boolean
    - users: list of dicts (rows) each corresponding to a user. Attributes:
        - user_id
        - username
        - levelReached

✅ Get/check user level

•

```
GET read_user_level?username="[username]"
   i.e. GET /read_user_level?username=test1
Return (json)
{
   "levelReached": [1 or 2 or 3],
   "username": [username] (the one you entered),
   "message": [info about result i.e. failure+reason or just success]
}
```

- •Usage: Frontend ChooseLevel page.
    - Get the integer value by accessing the key “level”
    - Will be used in Connor’s switch statement to determine which levels are disabled
- (i.e. frontend enforces that players can’t chose from 2 if they only completed level 1)

✓ Increment user level

When the user complete’s a level successfully POST to this method!

(See LevelSuccess use case in our use case document)

•

```
POST /increment_user_level?username=[username]

i.e. POST /increment_user_level?username=jatcs
```

•NOTE: the value of the level only changes if they hadn’t yet reached the final level (3).

- Use “GET /read_user_level” to verify the value of this increment occured as you wished :)

✓ Update user score

●

```
POST increment_score
Params:
- username
- score
```

- •score is the score they received on the current level, that is what we will add on to the current score. Basically, you don't have to remember what the previous score was.
- Usage: Frontend LevelComplete page (which gets this information from PlayLevel)

✅ Load leaderboard table

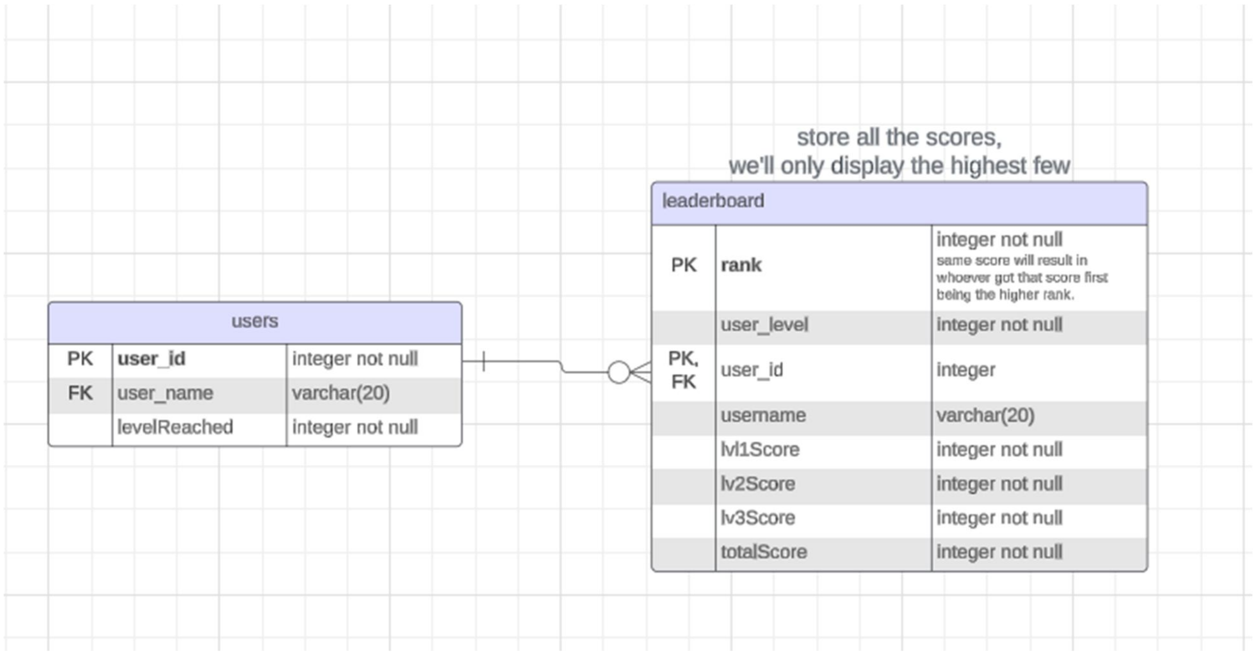
●

```
GET /load_leaderboard
- No params required
- Return: A list of rows to display (example). Currently it returns all the score results.
{
   "rows": [
      {
         "rank": 1,
         "username": "jat101",
         "score": 0
      }
   ]
}
```

- •Usage: Frontend ViewLeaderboard page.

Database

Samuel Oyeneyin



store all the scores,
we'll only display the highest few

users

PK	**user_id**	integer not null
FK	user_name	varchar(20)
	levelReached	integer not null

leaderboard

PK	**rank**	integer not null same score will result in whoever got that score first being the higher rank.
	user_level	integer not null
PK, FK	user_id	integer
	username	varchar(20)
	lvl1Score	integer not null
	lv2Score	integer not null
	lv3Score	integer not null
	totalScore	integer not null