GDD for COMP3016
Coursework 1
Portraits of a Vampyre

Vid: https://youtu.be/pM0SJI7M9NE

Gameplay:

The game is three rounds.
Round 1 starts with three people, and following each round, it adds two more people, so round 2 is five people and round 3 is seven people.
You must let most people in. Kicking out humans will lower your score.
If you let a vampire in, the game does not end.

At the end, you are given a score on how well you did. Keep count of the number of vampires you let in, and the number of humans you left out/kicked.

The idea is you gotta deduce if the person you are speaking to is a vampire or an actual human. The core loop is trying to figure out who is a vampire, but the game becomes a mess, as at no point are you told whether you are on the right path or not. It's about trying to figure out the vampires from the humans and getting a high score at the end.

Dependencies used:

- SDL2
- SDL2_image
- SDL2_ttf
- SDL2_mixer
- nlohmann _json

Use of AI description:

The use of AI was only for assisting with complex debugging, where I couldn’t find resources. I also used it for research to find resources for SDL2, such as Lazy Foo’s production beginning game programming v2.0.

Game programming patterns:

State:

```
enum class Scene{
   Start,
   RoundIntro,
   Gameplay,
   Score
};
```

———————

```
switch (currentScene) {
   case Scene::Start:
break;
case Scene::RoundIntro:
```

———————

I used the State Pattern. The code above controls the flow of the game. Starting with Start, RoundIntro, Gameplay and score. These allow loading assets and controlling the game in a manageable way.

Service locator:

```
Game::Game() : window(nullptr),
renderer(nullptr),
running(false),
KickButtonTexture(nullptr),
LetInButtonTexture(nullptr),
TextBoxTexture(nullptr)

{}
```

———————

```
guests = loadGuests(
       std::string (ASSETS_PATH)+"/data/guestList.json",
                   std::string
                   (ASSETS_PATH)+"/data/vampire_traits.json",
                   totalGuests, vampireCount
                   );
```

———————

```
target_compile_definitions(${PROJECT_NAME} PRIVATE
ASSETS_PATH="${CMAKE_SOURCE_DIR}/assets")
```

———————

In my game.cpp I located assets. With the JSON files, I used the nlohmann_json dependency. Due to issues with the code at the start of development, I had to set up a definition to call the JSON files.

Component:

The game is built in a way that separates the core responsibilities across classes.
Game.cpp, Rounds.cpp, guests.cpp
This was due to wanting to make the game manageable and readable.

Flyweight:

```
std::unordered_map<std::string, SDL_Texture*> textureCache;

for(auto& g : guests){
   //testing if I am reusing because I am crying
   if(textureCache.find(g.portraitPath) != textureCache.end()){
       g.texture = textureCache[g.portraitPath];
       continue;
   }
```

——----------

I took the idea of the pattern flyweight to optimise the portrait loading. Portraits are loaded once and stored. So, if an NPC is chosen at random to have the same portrait, they reuse the portrait instead of loading a copy of the portrait.

Game mechanics:

Player Interaction:

The main mechanic has been given the option to either let a person in or kick them out. This will be done by a mouse click after the user reads their information or perhaps glances at it.

```
if(event.type == SDL_MOUSEBUTTONDOWN && event.button.button ==
SDL_BUTTON_LEFT){
   int x = event.button.x;
   int y = event.button.y;

   if(x >= kickButtonReact.x && x <= kickButtonReact.x + kickButtonReact.w
       && y >= kickButtonReact.y && kickButtonReact.y + kickButtonReact.h)
   {
       std::cout<< "kick pressed\n";
       if(currentRound) currentRound->nextGuest(false);
   }
   if(x >= letInButtonReact.x && x <= letInButtonReact.x + letInButtonReact.w
      && y >= letInButtonReact.y && letInButtonReact.y + letInButtonReact.h)
   {
       std::cout<< "let in  pressed\n";
      if(currentRound) currentRound->nextGuest(true);
   }
```

I have assets and drawn rects in game.cpp.

Randomiser and guest loading:

Each launch of the game, the vampires are chosen at random along with portraits and names. Allowing for fresh replayability of the game.
Even the vampire is chosen at random

```
for (auto &item: guestJson["guests"]) {
   Guest g;
   g.id = item["id"];
   g.name = item["name"];
   g.gender = item["gender"];
   g.traits = item["traits"].get<std::vector<std::string>>();
   g.isVampire = false;

   if (g.gender == "m")
       g.portraitPath = getRandomPortrait(std::string(ASSETS_PATH) +
"/textures/M");
       //g.portraitPath = getRandomPortrait("assets/textures/M");

   else if (g.gender == "f")
       g.portraitPath = getRandomPortrait(std::string(ASSETS_PATH) +
"/textures/F");

   allGuests.push_back(g);
```

The asset folder is loaded, but to keep names and portraits in line. Like characters with male names are listed as ‘m’. This is called in the code above to assign a male painting.

```
std::ifstream vampireFile(vampireFilePath);
   json vampJson;
   vampireFile >> vampJson;
   vampireTraits = vampJson["vampireTraits"].get<std::vector<std::string>>();

   srand(static_cast<unsigned int>(time(nullptr)));
   for (int i = 0; i < vampireCount && !selectedGuests.empty(); i++) {
       int index = rand() % selectedGuests.size();
       selectedGuests[index].isVampire = true;

       for (int j = 0; j < 2; j++) {
           int tIndex = rand() % vampireTraits.size();
           selectedGuests[index].traits.push_back(vampireTraits[tIndex]);
       }
   }
   return selectedGuests;

}
```

The code above loads ‘vampire_traits.json’. They are labelled a vampire and given the vampire traits. This isn’t perfect, but it assigns random vampire traits to a guest at random

Score and endings:

Depending on how well the player does in the game, they will get four different endings.

```
std::string Rounds::getEndingMessage() const {
   //this for the end scene
   if (vampiresLetIn >= 3 ){
       return "Iron.. The smell of blood reeks the halls, a blood bath.\n You
let too many vampires in and no humans, \neven yourself, do not wake to see
the daylight. ";
   }else if( humansKicked >= 6){
       return "You played too safe, shame those turned away will have fate
undecided for better or worse";
   }else if( vampiresLetIn == 0 && humansKicked ==0){
       return "Hero. With every human saved and every vampire left outside,
\n you managed to let everyone see dawn with no bloodshed. \ncongrats";
   } else if(humansLetIn==0 && vampiresLetIn==0){
       return "at the end you hear a knock. Peeping through the hole of the
day \n you are overcome to let IT in \nbefore you know it grabs your hand,
pulls you up,\nand suck every drop of blood.\ndead and failure.";
   }else{
       return "Some humans are saved... while some lie outside not moving.";
   }

;}
```

These endings are tied to how well, the player performs throughout the game.

Round progression:

I added round progression. The idea is that after a certain amount of guests there will be a round buffer before the next guest is loaded..

```
case Scene::Gameplay:
   if (currentRound->isRoundOver()) {
       if (currentRound->getCurrentRound() == 3)
       {
           currentScene = Scene::Score;
           sceneStartTime = SDL_GetTicks();
       }
```

The code above checks if the round has gone through all three rounds, if so the scene score is loaded.

——----------

```
void Rounds::startNextRound(SDL_Renderer *renderer) {
   if(CurrentRound >= 3) {
       roundOver = true;
       return;
   }
   CurrentRound++;
   currentGuestIndex =0;
   roundOver= false;
   StartRound(renderer);

}
```

The code above increases the rounds and resets the guest index, preparing the next chunk of guests.

Scene flow:

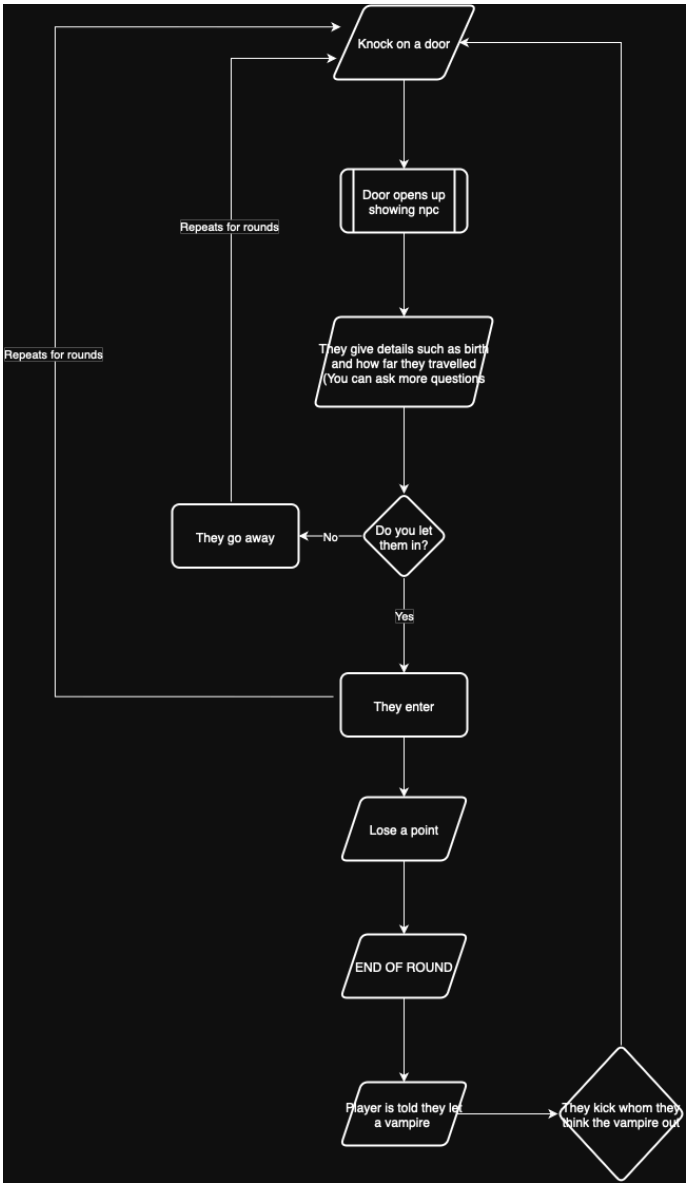
Mentioned in programming patterns. I used scenes to load parts of the game. They are timed because I thought it helped with pace.

The scenes were:

- Start
- Round intro
- Gameplay
- Score

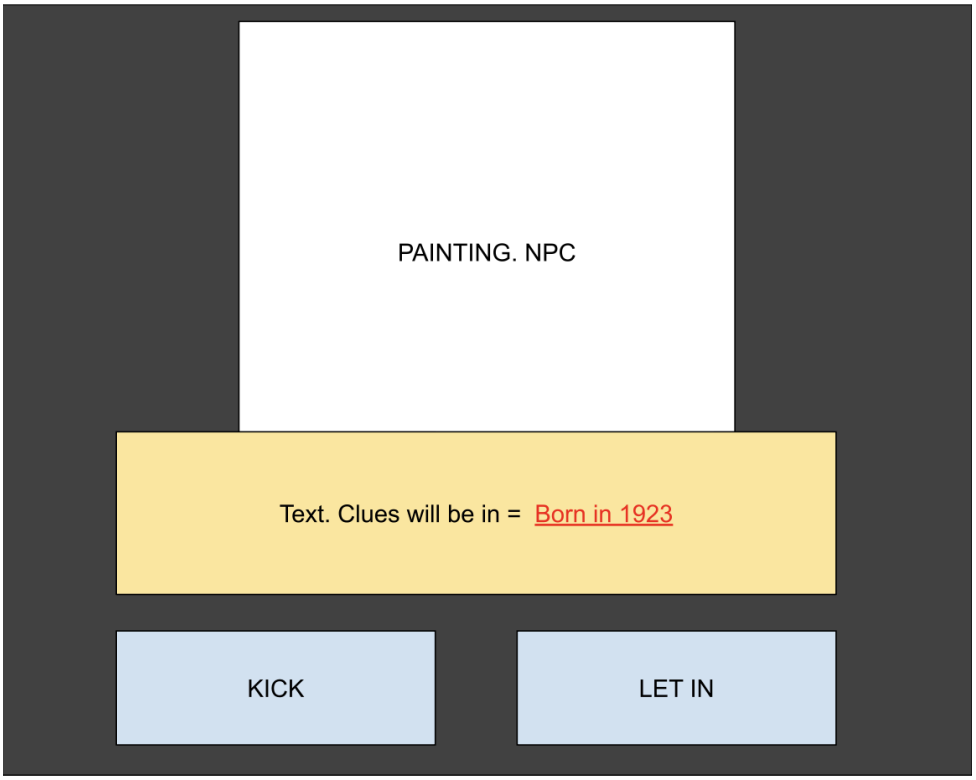
Round intro loads the intro, while the gameplay scene allows interactions with buttons.

UML Design diagram

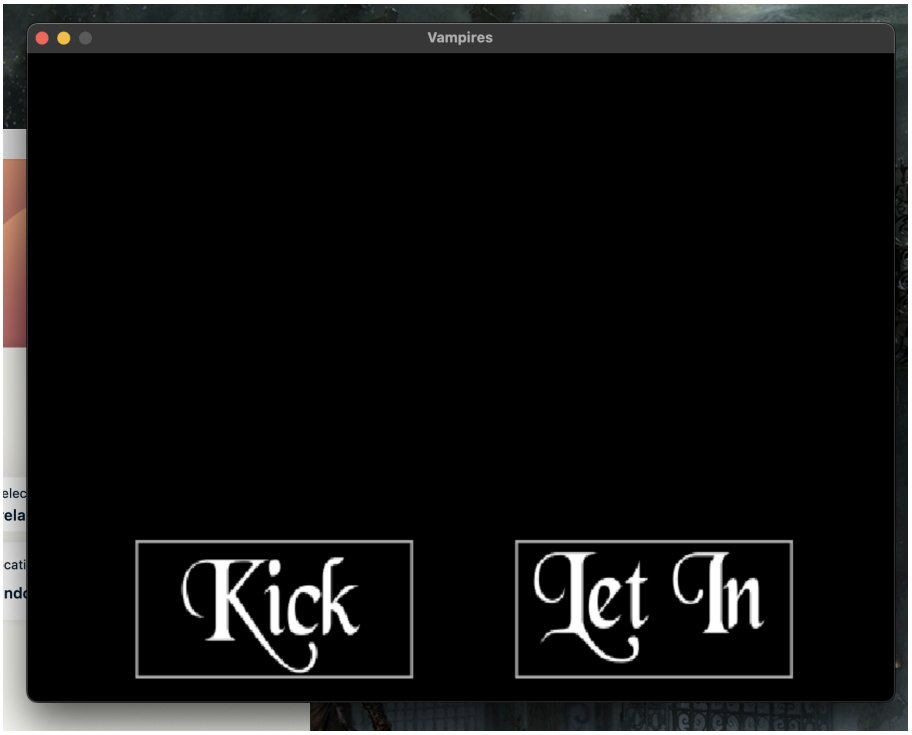


Sample Scenes:

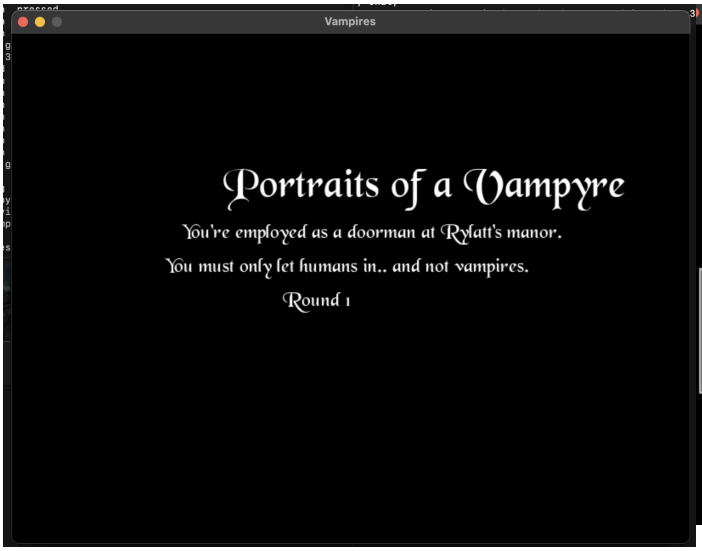
First draft:

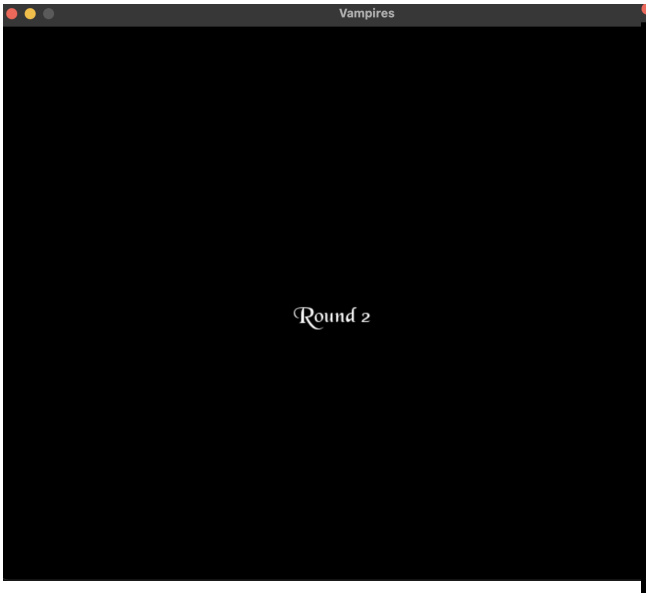


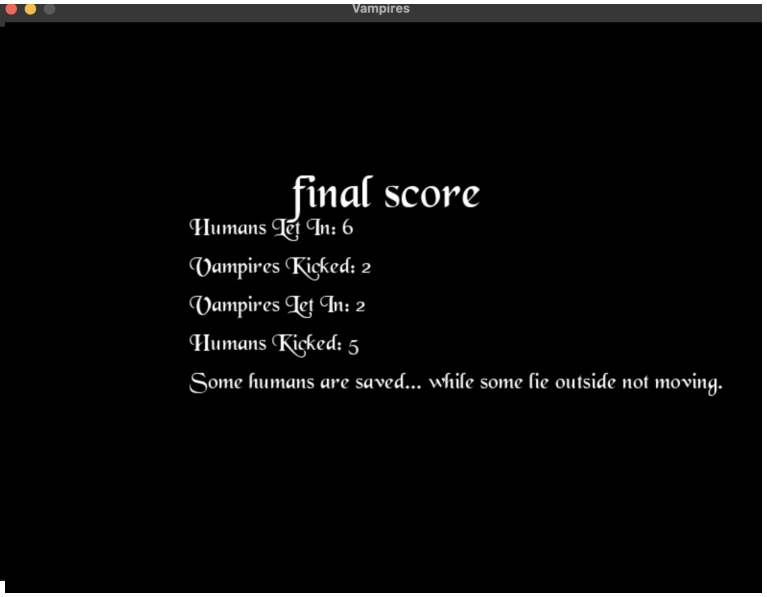
Second draft:



Final screens:







Expectation handling and test cases:

For a lot of the code, I used tests to check if assets and parts of the game were loading. Due to the nature of games with assets and JSON files, I had a lot of issues, so using small tests that give warnings helped the debugging process a lot.

Here are some examples, starting with loading assets. I made it print it out if it didn't load.

```
tempSurface = IMG_Load("../assets/textures/LetInButton.png");
if (!tempSurface) {
    std::cerr << "failed to load let in button: " << IMG_GetError() <<
std::endl;
} else {
    LetInButtonTexture = SDL_CreateTextureFromSurface(renderer, tempSurface);
    SDL_FreeSurface(tempSurface);
}
//textbox
tempSurface = IMG_Load("../assets/textures/BorderOutline.png");
if (!tempSurface){
    std::cerr <<"failed to load border: " << IMG_GetError() << std::endl;
}else
{
    TextBoxTexture = SDL_CreateTextureFromSurface(renderer, tempSurface);
    SDL_FreeSurface(tempSurface);
}
//font
font = TTF_OpenFont("../assets/fonts/BLKCHCRY.TTF", 24);
if(!font){
    std::cerr << "failed to load font: " << TTF_GetError() << std::endl;
}
//SMALL FONT& LARGE font
fontSmall = TTF_OpenFont("../assets/fonts/BLKCHCRY.TTF", 20);
fontLarge = TTF_OpenFont("../assets/fonts/BLKCHCRY.TTF", 50);

if(Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2,2048)<0){
    std::cerr<< "SDL_mizer cannot initalize: " << Mix_GetError() << std::endl;
}
rainMusic = Mix_LoadMUS("../assets/sounds/calming-rain.wav");
if (!rainMusic){
    std::cerr << "failed to load rain: " << Mix_GetError()<< std::endl;
}else{
    Mix_PlayMusic(rainMusic, -1);
    Mix_VolumeMusic(3);
}
```

I used the terminal to check other parts of the game as well such as guests, rounds.

```
std::ifstream test(g.portraitPath);
if(!test.good()){
    std::cerr << "portrait file not found: " << g.portraitPath<<std::endl;
    continue;
}
```

```
SDL_Surface* tempSurface = IMG_Load(g.portraitPath.c_str());
if(!tempSurface){
   std::cerr<< "failed to load portrait: " << g.portraitPath << " | "
<<IMG_GetError() << std::endl;
   continue;
}
g.texture = SDL_CreateTextureFromSurface(renderer, tempSurface);
SDL_FreeSurface(tempSurface);
if(!g.texture){
   std::cerr << "texture creation has failed for: " << g.portraitPath << " |
" << SDL_GetError()<<std::endl;
}
```

—----

A lot of the testing of the game was working. The error testing was trial and error. Loading the game and checking endings, and making sure different guests, vampires, were loading.

How the prototype works:

It's rather simple. You load vampires from cmake_build_debug.

The game is built so guests are loading from a JSON file. I did this so it would be easier to add more guests in the future.
The game also loads 'Vampire_traits.json'. This file has a list of vampire traits. The game at random will randomly pick a guest to be labelled a vampire, and they will be assigned two vampire traits.
Portraits are also assigned to a guest at random, although paintings are gendered, and so are the guests.

The gameplay is really just pressing the kick button, which kicks the guest out, or Let In, which lets the guest in.
Each guest is timed for 60 seconds.

The game tracks how many guests you have let in/kicked, and how many vampires you have let in/kicked.

Depending on your score. You will get a different ending.

There is a rain ambience I added that is stock-free.

Evaluation:

Being honest, I feel a mixture of a sense of pride but a sense of being worryness. After last year, I was disappointed with the work I submitted. I felt like I had disappointed myself.

So in the summer, in the middle of my part-time job, I started to learn C++ and make an effort of learning by reading ‘Game Programming Patterns’, and ‘Game Engine Architecture’.

When it comes to the project, I had an idea of how I would do this, and I feel like, for the most part, I did what I wanted to do, but it could have been more fleshed out.
Like preloading sample texts that the traits will go into.

If I did this project differently, I would have liked to get custom assets. I do like the idea of using copyright-free paintings, but I do think the game would have been aesthetically better if I used my own assets.