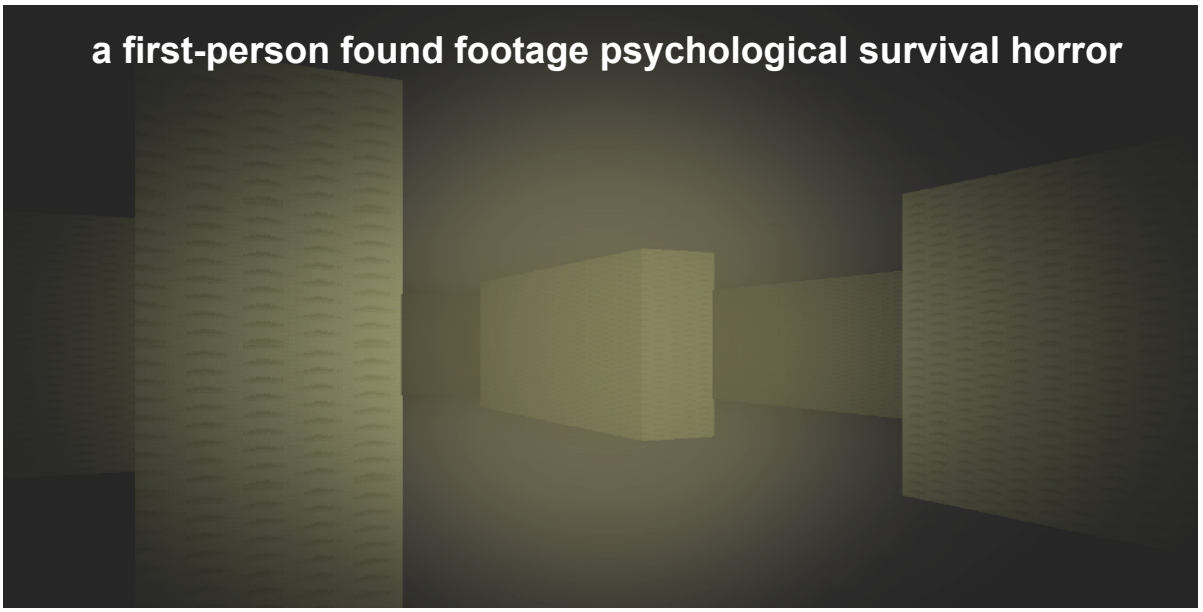
Maciej Kozłowski, Jakub Stacherski, Krzysztof Pałucki, Adam Jeż

BACKROOMS



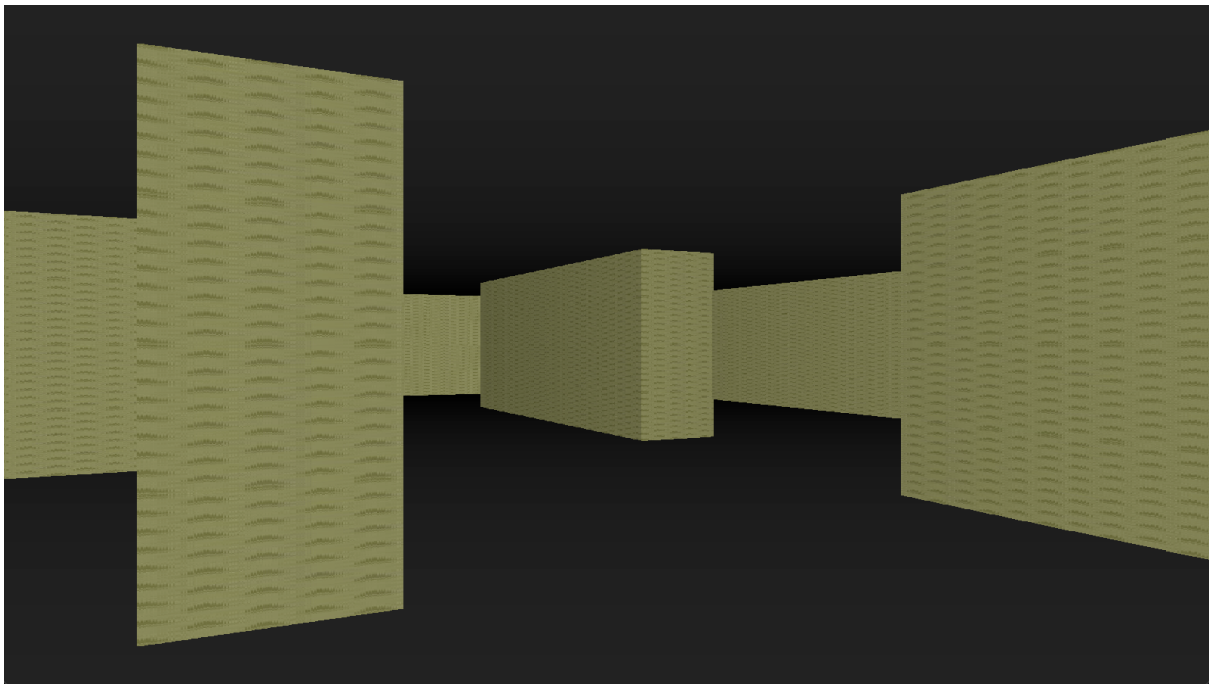
Cel projektu:

Celem projektu było stworzenie gry horror 3D, wykorzystującą dostępne biblioteki do C++ *“Box2D”* oraz *“SFML”*. Gra w swoim założeniu ma być prezentacją zastosowania technologii renderowania obrazu 3D raycasting.

Opis:

W grze znajdujemy się w losowo generowanym labiryncie. Celem gry jest zdobycie jak największej liczby karteczek (dalej “Page”), jednocześnie unikając bezpośredniego kontaktu z potworem (dalej “Enemy”), który za nami podąża, jeśli jesteśmy w jego polu widzenia. Obecnie posiadaną liczbę można sprawdzić w trybie debugowania (dalej “Debug mode”). Dodatkowo, można wyposażyć się w detektor (dalej “Emf”), posiadający trzy tryby informowania o naszej odległości od “Enemy” za pomocą cyfr, które się podświetlają na dany kolor(1 - bezpieczna odległość, a 3 - potwór bardzo blisko). Gra kończy się w momencie, gdy “Enemy” zderzy się z naszą postacią (mamy uniemożliwione poruszanie się).

Zdjęcie 1: “Enemy” i jego wpływ na tryb wykrywania w urządzeniu “Emf”.



Zdjęcie 2: Widok po włączeniu “Debug mode”

SPOSÓB TESTOWANIA:
Ze względu na charakter projektu, wiele elementów zostało przetestowanych “na żywo”, podczas uruchamiania programu. Tylko niektóre mechaniki mają dodane testy jednostkowe. Testy są kompilowane poprzez komendę *“make”* lub bezpośrednio *“make tests”*. (będąc w folderze *“backrooms”*) Uruchamiane są komendą *“./bin/tests.exe”.*

SPOSÓB INSTALACJI:
a) dla *Windowsa*:
Przed uruchomieniem zainstalować środowisko Mingw32. Skompilować komendą *[Location of mingw32-make.exe]/mingw32-make.exe*
b) dla *Linuxa*:
Zainstalować bibliotekę sfml za pomocą komendy w konsoli *”sudo apt-get install libsfml-dev”.* Kompilować komendą *“make”*.’
c) dla MacOS:
Zainstalować bibliotekę sfml i Box2D korzystając z menedżera pakietów Homebrew za pomocą komendy w konsoli “brew install sfml” i “brew install box2d”. Bibliotekę box2d należy potem zmodyfikować zgodnie z opisem w pliku README.md w folderze projketu _MACOS.
Uruchamianie:
“./bin/main.exe”

OBSŁUGA:
- *WSAD* - poruszanie się
- *E* - podnoszenie przedmiotu (“Emf” lub “Page”)
- *G* - upuszczanie przedmiotu
- *Q* - zmiana przedmiotu znajdującego się na wyposażeniu z ekwipunku
- *shift* - bieganie
- *slash* - włączanie “Debug mode” (“Enemy” nie goni nas, brak widocznej mgły, można zobaczyć tryb chodzenia “Enemy” losowo po labiryncie, w konsoli wyświetla się ilość obecnie podniesionych “Page”, w momencie, gdy patrzymy bezpośrednio na kolejną kartkę)

Krótki opis zaimplementowanych klas:

- **Camera** - Tworzy obraz widziany oczami gracza za pomocą techniki raycast. Nie jest odpowiedzialna za wyświetlanie go na ekranie.
- **Chunk** - Reprezentuje fragment mapy, na którego składają się 2 ściany - północna i zachodnia. Dzięki temu, po złożeniu wielu chunków obok siebie, możemy otrzymać siatkę ścian.
- **Component** - Wirtualna klasa służąca do implementacji wzorca mediatora.
- **DrawableObject** - Obiekt, który posiada teksturę. Można go narysować na ekranie.
- **Emf** - Przedmiot wykrywający przeciwnika.
- **Enemy** - Przeciwnik podążający w stronę gracza. Gdy gracz jest niewidoczny, idzie w miejsce, gdzie ostatnio go widział. Gdy dojdzie w to miejsce, zaczyna chodzić losowo, jednak nie uderzając w żadną ścianę.
- **GameState** - klasa przechowująca informację o aktualnym stanie gry. Jej atrybutami są Player i Enemy. Jest mediatorem pośredniczącym w komunikacji między obiektami.
- **Image** - Przechowuje wszystkie obiekty sf::Image.
- **Item** - Klasa wirtualna. Reprezentuje obiekt leżący na ziemi, który można podnieść będąc w pobliżu.
- **Mediator** - Klasa wirtualna. Służy do implementacji wzorca mediatora.
- **MyListener** - Klasa dziedzicząca po b2ContactListener. Obsługuje zderzenia między obiektami: np po zderzeniu Enemy z Player'em, zabija Player'a.
- **Object** - Reprezentuje obiekt obecny w świecie gry. Klasa bazowa dla wielu innych obiektów.
- **Object2D** - Obiekt wyświetlany jako płaska tekstura, zawsze zwrócona w stronę gracza.
- **Object3D** - Obiekt trójwymiarowy, wyświetlany z perspektywą.
- **Page** - Przedmiot (kartka) do zbierania, ze statycznym licznikiem zebranych kartek.
- **Player** - Reprezentuje gracza. Porusza się za pomocą inputu z klawiatury i myszki.
- **RandomGenerator** - generator losowych liczb. Pozwala na implementację proceduralnego generowania mapy.
- **Ray** - klasa umożliwiająca wysłanie promienia w zadanym kierunku i odczytanie miejsca zderzenia z potencjalnym obiektem.
- **Textures** - Przechowuje wszystkie obiekty sf::Texture
- **Timer** - klasa umożliwiająca mierzenie upływu czasu w grze.
- **UserIO** - klasa zbierająca input z klawiatury i myszki oraz pozwalająca narysować texturę w odpowiednim miejscu na ekranie. Jej atrybutem jest sf::RenderWindow.
- **World** - klasa przechowująca wszystkie chunki i dbająca o ich generowanie oraz usuwanie.

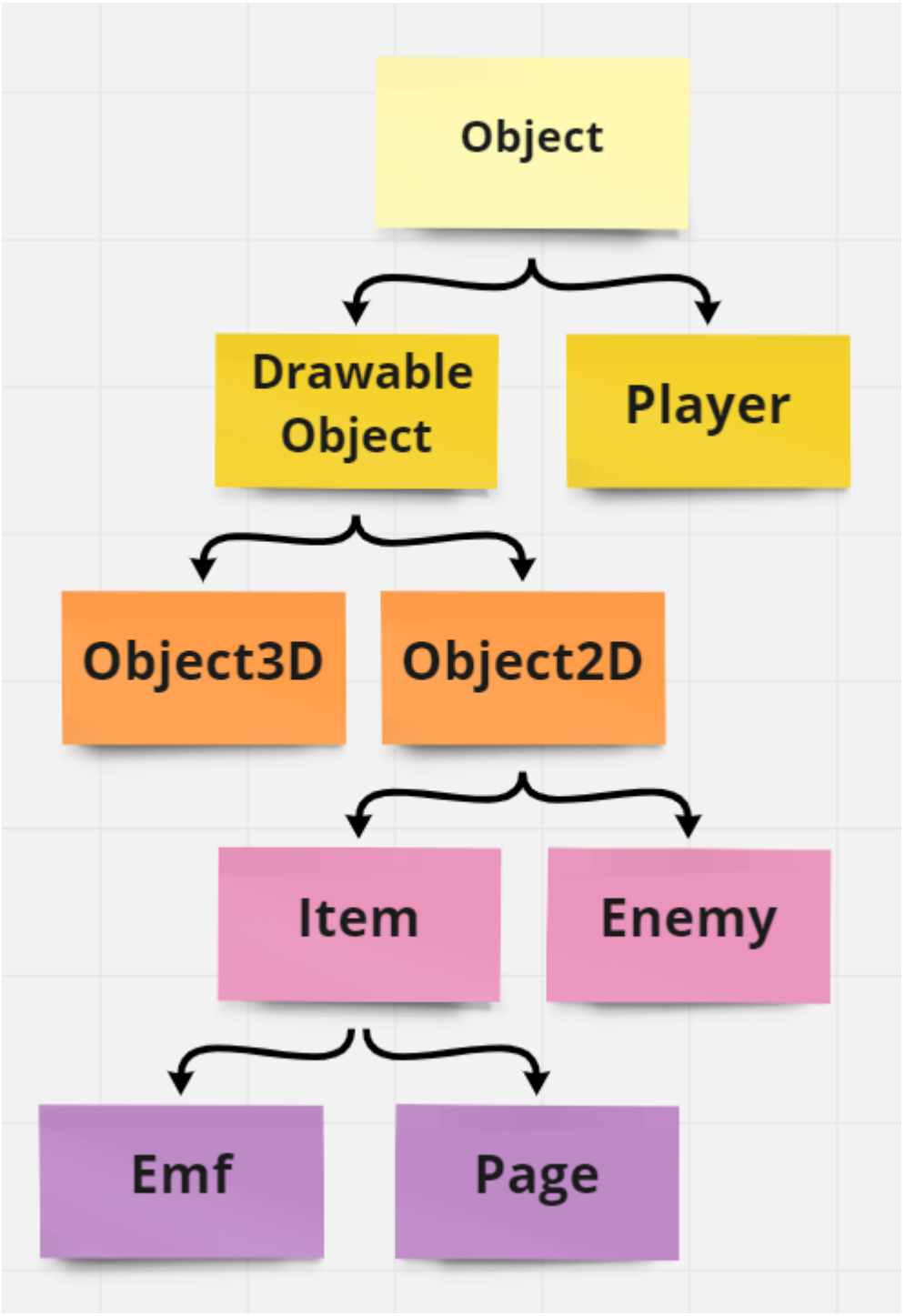
Wiele metod zostało opisanych bezpośrednio w kodzie.

PARADYGMAT OBIEKTOWY:

W projekcie zastosowane zostały wszystkie paradygmaty programowania obiektowego. Występują klasy wirtualne, dziedziczenia (na rysunku “Hierarchia klas”)

abstrakcja - mediator

Hierarchia klas:



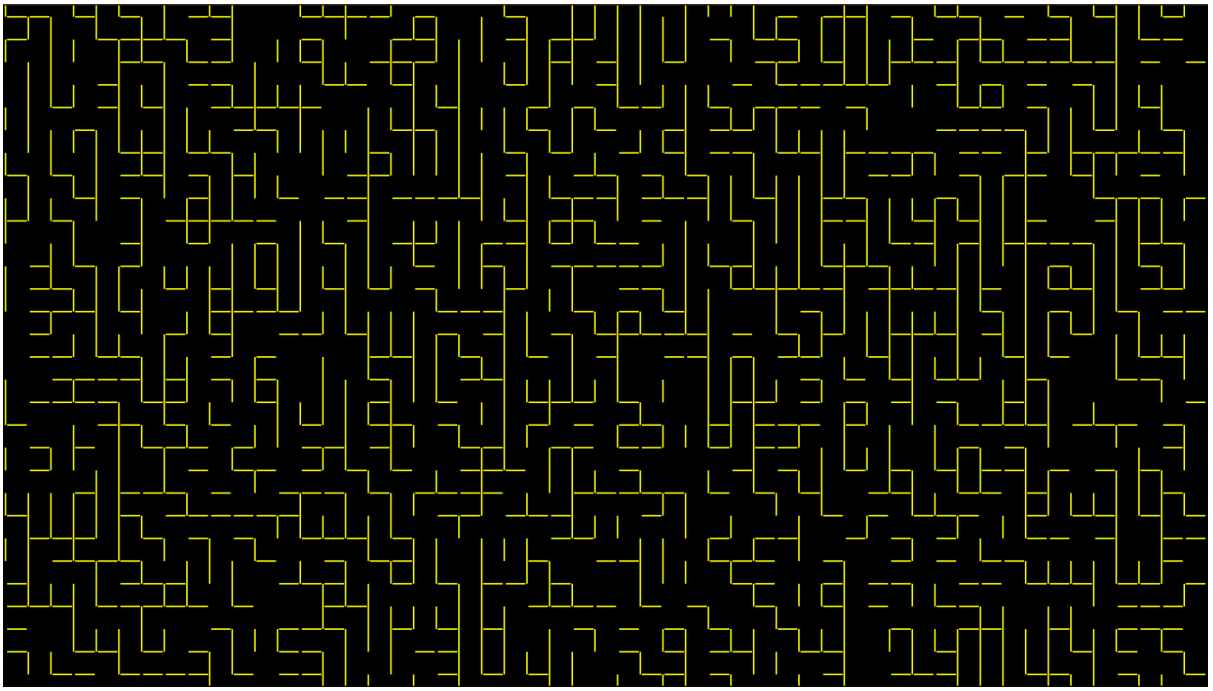
Rys. 1: Hierarchia klas w projekcie

WYZWANIA:

Ze względu na skalę projektu oraz mnogość pomysłów, podczas etapu tworzenia, wyzwania stanowiły nieodłączną jego część. Aby im sprostać, w projekcie zostało zaimplementowanych wiele ciekawych rozwiązań programistycznych. Znacząco podnoszą one poziom zarówno zaawansowania gry, jak i immersji podczas seansu. Spośród wielu z nich można wyróżnić:

1) **tworzenie obrazu 3D (raycasting):**

   Promienie wychodzące z kamery przecinają się z obiektami umieszczonymi w świecie. Na podstawie kąta padania tego promienia na punkt określana jest jego jasność. Im bliżej ten punkt znajduje się kamery
2) **wzorce projektowe:**
   a) mediator
   b) builder
3) **generowanie proceduralne:**
   a) zachowanie obiektu klasy “Enemy”
   b) tworzenie labiryntu:



Zdjęcie 3: WIdok z góry wygenerowanego labiryntu