**Brick Breaker – dokumentacja projektu**

Gra w C++ z użyciem biblioteki SFML – Kamil Skop, Julian Granieczny, Michał Kosiec

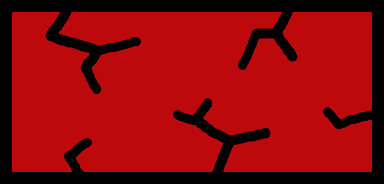
1. Ogólny opis zawartości i funkcjonalności

a) Wprowadzenie

Gra Brick Breaker jest okienkową grą, przeznaczoną na systemy Windows (z możliwym późniejszym wyeksportowaniem na Linux i inne systemy operacyjne, wspierane przez SFML). Rozgrywka polega na odbijaniu piłeczki paletką i niedopuszczeniu do jej spadnięcia, przy czym kąt odbicia zależy od miejsca na paletce, na które piłeczka spadnie, a celem gracza jest pozbycie się z planszy wszystkich możliwych do zniszczenia cegiełek poprzez odpowiednie manewrowanie paletką. W grze występują różne typy cegiełek o różnych właściwościach. Podczas rozgrywki należy uważać na cykliczne obniżanie się planszy, ponieważ gdy jakakolwiek cegiełka dotknie paletki lub strefy unicestwienia piłeczek, natychmiast następuje przegrana. Gra zawiera 40 poziomów, każdy odblokowywany po przejściu poprzedniego, które starają się stopniowo wprowadzać różnorodne mechaniki przy równoczesnym zwiększaniu poziomu trudności. W grze zawarta jest również instrukcja obsługi, a także ustawienia do kontroli głośności dźwięku oraz muzyki.

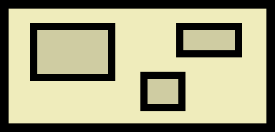
b) Spis występujących w grze cegiełek.

1. **Zwyczajna** – Cegiełka o 4 stanach zniszczenia. Każde uderzenie piłeczką powoduje coraz więcej pęknięć, aż w końcu cegiełka znika.

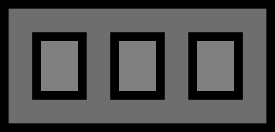
   Obraz zawierający czerwony, Grafika, sztuka, design

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

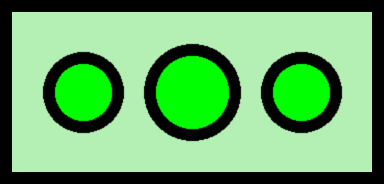
1. **Wrażliwa** – Cegiełka, która znika po jednym uderzeniu. Nie odbija piłeczki, więc ta przez nią przelatuje.



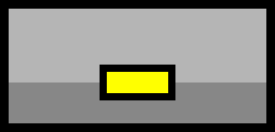
1. **Niezniszczalna** – Cegiełka, która nie może zostać zniszczona. Służy za element konstrukcyjny w niektórych poziomach.



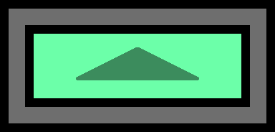
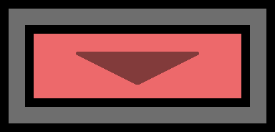
1. **Powielająca** – Cegiełka, która po swoim zniszczeniu powiela piłeczkę, która w nią uderzyła. Z jednej piłeczki powstają trzy.



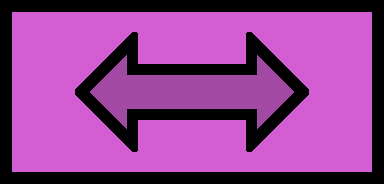
1. **Powiększająca** / **Pomniejszająca** – Cegiełka, która po swoim zniszczeniu powiększa / pomniejsza paletkę. Jej przeciwieństwa redukują się.

1. **Przyspieszająca** / **Spowalniająca** – Wariant cegiełki niezniszczalnej, który przyspiesza / spowalnia piłeczkę po jej odbiciu.

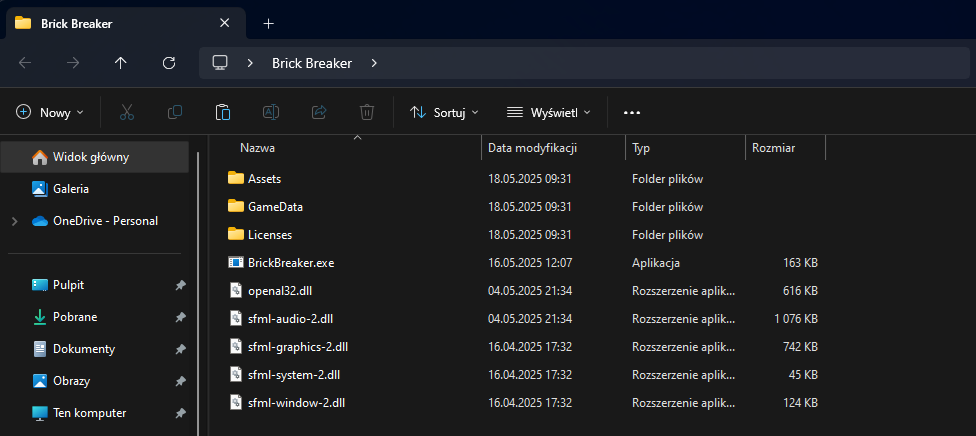
1. **Odwracająca** – Cegiełka, która po swoim zniszczeniu odwraca sterowanie lub, gdy już jest odwrócone, przywraca domyślne.



2. Instrukcja użytkownika

a) Uruchamianie gry

Aby, uruchomić grę, należy na systemie Windows wejść w link <https://github.com/Kamiloso/BrickBreaker/releases> oraz pobrać plik .zip dla najnowszej wersji. Następnie należy wypakować wszystkie pliki do jednego folderu i uruchomić plik wykonywalny .exe znajdujący się wewnątrz. Nie należy uruchamiać gry bez wcześniejszego wypakowania zawartości ani też modyfikować rzeczy, znajdujących się w katalogu gry.





**Uwaga:** System Windows może wyświetlić monit, blokujący aplikację ze względu na jej nierozpoznanie. Wtedy należy kliknąć w tekst Więcej informacji, a następnie „Uruchom mimo to”.

Obraz zawierający tekst, zrzut ekranu, Jaskrawoniebieski, System operacyjny

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna. 



b) Instrukcja użytkowania

Po uruchomieniu użytkownik znajdzie się w menu głównym. Należy wybrać LEVELS, a następnie pierwszy poziom (później wyjaśnione zostanie, jak można w ramach testów odblokować je wszystkie bez ich przechodzenia). W zakładce INSTRUCTION znajduje się skrócony opis instrukcji oraz opcje regulacji głośności dźwięku.

Obraz zawierający tekst, zrzut ekranu, Czcionka, miejsce parkingowe/przestrzeń

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Obraz zawierający zrzut ekranu, kwadrat, Wielobarwność, Prostokąt

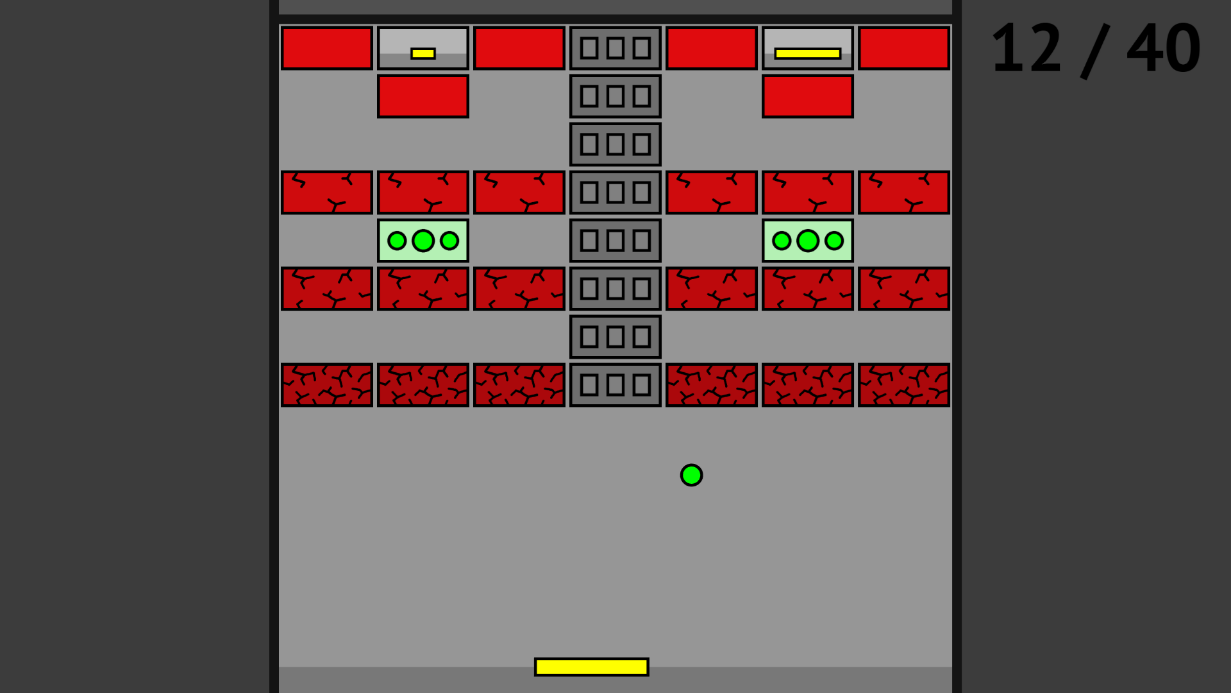
Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Rozgrywka polega na odbijaniu piłeczki za pomocą paletki, którą przemieszczać można po dolnej części planszy za pomocą strzałek. Kąt odbicia piłeczki zależy od odległości od środka paletki, na którą spadnie piłeczka. Celem rozgrywki jest zniszczenie wszystkich cegiełek, które da się zniszczyć, przy czym nie należy dopuścić do utraty piłeczki, ani zejścia cegiełek do poziomu paletki (spowoduje to przegraną).

Sterowanie:

1. Strzałki 🡨 i 🡪 lub klawisze **A** i **D** – poruszanie paletką
2. **F11** lub **F** – zmiana między trybem pełnoekranowym, a okienkowym
3. **ESC** – pauza, anuluj (w zależności od kontekstu)

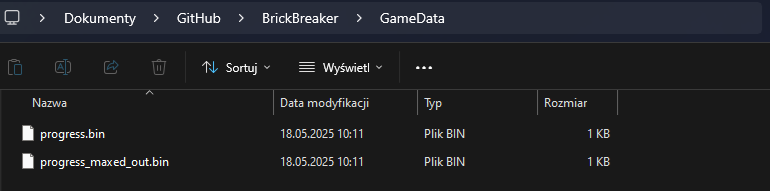
Poniżej znajduje się zrzut ekranu, przedstawiający jeden z poziomów.



Długoterminowym celem gracza jest odblokowanie i przejście wszystkich poziomów. Postęp jest zapisywany między sesjami gry lokalnie w folderze z grą.

c) Natychmiastowe odblokowanie wszystkich poziomów

Istnieje ukryta możliwość odblokowania wszystkich poziomów, przeznaczona do testów oraz dla osób, które nie mają czasu na grę, ale chcą obejrzeć projekt (np. wykładowcy). Aby z niej skorzystać, należy w plikach gry przejść do folderu *./GameData/* a następnie przemianować znajdujący się tam plik *progress\_maxed\_out.bin* na *progress.bin*. Jeśli plik *progress.bin* już istnieje, należy go stamtąd usunąć. Procedura ta odblokuje wszystkie poziomy i zresetuje zmiany w ustawieniach dźwięku.



3. Opis projektu od strony technicznej

a) Szczegóły techniczne, dotyczące projektu i kompilacji.

Aby poprawnie zbudować grę, należy pobrać projekt z linku <https://github.com/Kamiloso/BrickBreaker/releases>, upewnić się, że używana jest konfiguracja zgodna z tabelą poniżej (w razie potrzeby/problemów zmienić), a następnie uruchomić kompilację w trybie Release. Po skompilowaniu, należy uruchomić skrypt *A-BuildFinalize.bat* znajdujący się w głównym katalogu projektu, który przeniesie wszystkie wymagane zasoby do docelowego folderu *./Release* oraz usunie z niego pliki *BrickBreaker.pdb* oraz *GameData/progress.bin*. Tak przygotowany folder *./Release* staje się pełnoprawnym buildem gry. Należy pamiętać, ze po każdym uruchomieniu gry utworzony zostaje plik *GameData/progress.bin*, który trzeba usunąć po przetestowaniu działania.

|  |  |
| --- | --- |
| **Nazwa projektu** | Brick Breaker – Release 1.0 |
| **Język programowania** | C++14 |
| **Środowisko programistyczne** | Visual Studio 2022 (kompilator MSVC 19.41) |
| **Biblioteka graficzna** | SFML 2.6.2 |
| **Architektura** | Windows x86 (32-bit) |

b) Struktura projektu.

W głównym katalogu projektu znajdują się wszystkie skrypty, natomiast po wejściu w plik *BrickBreaker.sln* zostają one uporządkowane w widoku Visual Studio za pomocą używanych tam tzw. filtrów. Zasoby, potrzebne do zaimportowania do finalnego builda umieszcza się w folderze *./Assets*, licencje w folderze *./Licenses*, a folder *./GameData* przeznaczony jest na przechowywanie danych przez grę między sesjami. Wszystkie te trzy foldery są przenoszone do folderu *./Release* podczas wykonywania się skryptu *A-BuildFinalize.bat*, przy czym z ostatniego usuwany jest również plik *./GameData/progress.bin*.

c) Struktura kodu

Kod gry jest niemal całkowicie oparty na paradygmacie obiektowym. Przy jego tworzeniu nieustannie korzystano z klas, wraz z mechanizmami dziedziczenia i polimorfizmu. Struktura skryptów jest silnie hierarchiczna. Przyjęto konwencję, że klasy dziedziczące z jednej klasy bazowej znajdować się będą w wewnątrz tego samego filtra (folderu w projekcie VS). Istnieją w projekcie dwa pliki nie obiektowe:

* main.cpp – Odpowiedzialny za start programu, zawiera funkcję main().
* common.h – Zawiera stałe globalne, z których korzysta cała gra.

Poniżej przedstawiony został opis wszystkich pozostałych skryptów wraz z ich hierarchią w układzie filtrów. Każda klasa ma przypisany plik .cpp oraz .h o tej samej nazwie, co nazwa klasy.

1. Colliders – Zawiera hierarchię dziedziczenia colliderów, zarządzających kolizjami oraz odbiciami piłki z elementami na scenie. Collidery zakładają, że piłeczka jest punktem (mimo, że nie jest), a rekompensują to swoim ułożeniem na zewnątrz obiektów, od których piłeczka ma się odbijać. Dzięki temu było możliwe stworzenie realistycznych odbić od narożników.

* [BAZOWA] Collider.h/cpp – Klasa bazowa dla colliderów. Zawiera podstawowe funkcje, jakie powinien mieć collider, ale bez ich implementacji, tzn. *getTimeToCollision()* oraz *bounceBall()*.
* CircleCollider.h/cpp – Collider w kształcie okręgu.
* LineCollider.h/cpp – Collider w kształcie linii poziomej lub pionowej. Działa tylko od jednej, wybranej strony.
* PlateCollider.h/cpp – Specjalny collider, przypisany paletce. Odbicia piłeczki od paletki są na tyle specyficzne, że należało stworzyć dla nich osobną klasę.

1. Foundation – Zawiera klasę GameWindow.

* GameWindow.h/cpp – Klasa, nadzorująca wyświetlanie okna gry oraz zawierająca metody, wymagane z wielu różnych miejsc w programie. Pełni ona również funkcję managera scen. Istnieje tylko jeden obiekt tej klasy.

1. Management – Zawiera klasy statyczne, zarządzające różnymi aspektami projektu.

* Input.h/cpp – Klasa statyczna, udostępniająca interfejs dla łatwiejszego zarządzania systemem wejścia (klawiatura, myszka) z użyciem wbudowanych w bibliotekę SFML mechanizmów.
* LevelGetter.h/cpp – Klasa statyczna, zarządzająca wczytywaniem poziomów, a także zapisem i odczytem danych z pliku *./GameData/progress.bin*.
* Sound.h/cpp – Klasa statyczna, zarządzająca dźwiękiem i muzyką za pomocą biblioteki SFML.
* Texture.h/cpp – Klasa statyczna, odpowiedzialna za wczytywanie, a także wygładzanie tekstur z użyciem biblioteki SFML i własnych algorytmów.

1. Other – Zawiera klasy, nie pasujące do żadnej z opisanych tutaj kategorii.

* Particle.h/cpp – Klasa pomocnicza, reprezentująca pojedynczą cząsteczkę, używaną przez ParticleSystem.

1. SceneObjects – Zawiera klasy, należące do hierarchii dziedziczenia obiektów scenicznych. Tylko takie obiekty można umieścić na scenie i je wyświetlać.

* [BAZOWA] SceneObject.h/cpp – Klasa bazowa dla obiektu na scenie. Wyłącznie instancje klas dziedziczących po tej klasie mogą zostać przypisane do sceny. Posiada ona wirtualne metody takie jak: *earlyUpdate()*, *lateUpdate()*, *draw()*, *getEvent()*, *getEventToScene()* i inne*,* które mogą zostać nadpisane w dowolny sposób dla każdego obiektu dziedziczącego.

1. Bricks – Klasy poszczególnych rodzajów cegiełek.
   * [BAZOWA] Brick.h/cpp – Klasa bazowa dla wszystkich cegiełek. Zawiera metody wirtualne takie, jak *unbreakable()*, czy *getActionsOnDestroy()*, które mogą zostać nadpisane w cegiełkach pochodnych według własnych wymagań.
   * FragileBrick.h/cpp – Klasa cegiełki wrażliwej.
   * MovementBrick.h/cpp – Klasa cegiełki zmieniającej prędkość piłeczki.
   * NormalBrick.h/cpp – Klasa zwykłej, czerwonej cegiełki.
   * ReverseBrick.h/cpp – Klasa cegiełki, odwracającej sterowanie.
   * SizeBrick.h/cpp – Klasa cegiełki, zmieniającej rozmiar paletki.
   * SpawnBrick.h/cpp – Klasa cegiełki, powielającej piłeczkę.
   * UnbreakableBrick.h/cpp – Klasa cegiełki niezniszczalnej.
2. Game – Klasy bezpośrednio związane z mechanikami rozgrywki.

* Ball.h/cpp – Klasa piłeczki.
* Plate.h/cpp – Klasa paletki.

1. Other – Klasy inne, niemożliwe do zakwalifikowania do pozostałych kategorii.

* EffectBounce.h/cpp – Klasa, reprezentująca eksperymentalny, aczkolwiek finalnie wyłączony efekt wizualny odbić piłeczki.
* ParticleSystem.h/cpp – Klasa, reprezentująca system cząsteczek.
* Star.h/cpp – Klasa, reprezentująca migającą gwiazdkę w menu głównym.

1. Shapes – Klasy podstawowych kształtów. Dziedziczą po nich bardziej złożone obiekty sceniczne.

* Circle.h/cpp – Klasa okręgu z konturem (lub bez).
* Rectangle.h/cpp – Klasa prostokąta z konturem (lub bez).

1. UI – Klasy związane z interfejsem użytkownika

* Button.h/cpp – Klasa przycisku. Przycisk wykrywa kliknięcia na sobie, które za pomocą metod *setEvent()*, *getEvent()* oraz *getEventToScene()* można obsłużyć. Oprócz tego zmienia kolor w zależności od swojego stanu.
* Text.h/cpp – Klasa, używająca biblioteki SFML do renderowania tekstu za pomocą zaimportowanej wcześniej czcionki *PTSans-Bold.ttf*.

1. Scenes – Zawiera klasy, należące do hierarchii dziedziczenia scen.

* [BAZOWA] Scene.h/cpp – Klasa abstrakcyjna, reprezentująca podstawową scenę, w której przechowywane są obiekty, dziedziczące po SceneObject (obiekt na scenie). Scena stanowi centrum decyzyjne, dotyczące głównych mechanik gry. Mechanizmy zarządzania scenami znajdują się w klasie GameWindow.
* GameScene.h/cpp – Klasa, reprezentująca scenę gry. Odpowiada za zarządzanie głównymi mechanizmami rozgrywki tj. poruszanie piłeczkami, fizyka gry, sprawdzanie zwycięstwa itd. Przechowywane są w niej takie obiekty, jak paletka, cegiełki, ściany planszy itp.
* LevelMenu.h/cpp – Klasa nieużywana. Jest to scena, która pierwotnie miała przechowywać w sobie menu wyboru poziomów, ale ze względu na zmianę koncepcji straciła swoje zastosowanie. Obecnie nigdy się ona nie wyświetli.
* MainMenu.h/cpp – Klasa, reprezentująca menu główne. Zawiera w sobie przede wszystkim opis układu menu głównego i funkcjonalności, związane z interakcją z nim.

d) Opis i wyjaśnienie ruchu piłeczek

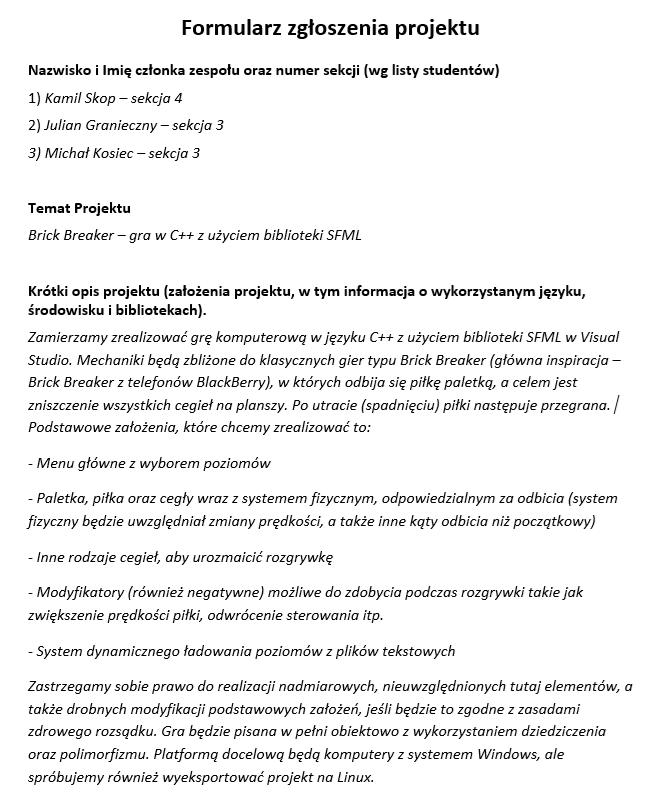
Fizyka piłeczek w grze może wydawać się trywialna, ale w rzeczywistości wymagała ona nieszablonowych rozwiązań i stworzenia dość zawiłego algorytmu. Po pierwsze, aby uniknąć problemów z odbiciami piłeczki od ostrych krawędzi cegiełek, zastosowano uproszczony opis matematyczny. Zamiast traktować piłeczki jako koła, postanowiono potraktować je jako punkty, a przy tym oddalić krawędzie colliderów od wszystkich obiektów o promień piłki. W powstałych lukach na narożnikach utworzono dopasowane collidery, będące okręgami. Okazuje się, że uderzenie środka koła w taki okrągły collider jest równoważne uderzeniu koła w narożnik, co daje naturalny efekt odbicia. Inną sprawą jest fakt, że piłeczki mogą poruszać się z dużymi prędkościami, a także może wystąpić więcej niż jedno odbicie w jednej klatce gry. W związku z tym, piłeczki w czasie jednej klatki poruszają się nie według kolejności, w której znajdują się w wektorze, ale na bazie tego, która z nich ma najkrótszy czas do uderzenia w jakikolwiek collider. Ponadto, piłeczka nie wykonuje całego swojego ruchu w klatce na raz. Po uderzeniu w collider, może przekazać kolejkę innym piłeczkom, które uderzyłyby w jakiś collider wcześniej. Oczywiście, wszystkie piłeczki w końcu wykonają pełny ruch ze swojej klatki, ale w takiej kolejności, że gwarantowane jest, że niezależnie od ilości klatek na sekundę, odbicia nastąpią w tej samej i poprawnej kolejności. Piłeczki nie mogą wylecieć poza mapę nawet przy ogromnych prędkościach, ponieważ przy kolizjach poruszają się one w sposób ciągły, a nie dyskretny.

4. Opis ciekawostek związanych z realizacją

* Jako, że biblioteka SFML udostępnia jedynie podstawowe funkcjonalności, związane z okienkowymi aplikacjami, konieczne było napisanie od zera wielu elementów, które w językach bardziej wysokopoziomowych niż C++ byłyby już zaimplementowane, przykładowo: menager scen, klasa przycisku, zaawansowana interpolacja tekstur.
* Podczas pracy z projektem używane były git w formie konsolowej, a także GitHub Desktop, w zależności od preferencji członka zespołu. Znacznie ułatwiło to wspólne tworzenie projektu i pozwoliło autorom na naukę obsługi tych narzędzi w praktyce.
* Gra Brick Breaker była inspirowana popularną grą z telefonów BlackBerry o tej samej nazwie. Z kolei ta, była najpewniej inspirowana jeszcze popularniejszą grą Arkanoid z roku 1986 lub Breakout z roku 1976. We wszystkich tych grach należy rozbijać cegiełki za pomocą piłki. Krótko mówiąc, stworzona gra nie jest niczym wybitnie oryginalnym. Zawiera natomiast również autorskie rozwiązania, jak np. specjalne cegiełki, zamiast których w klasykach znajdziemy spadające power-upy.

5. Podsumowanie i wnioski

a) Przypomnienie formularza zgłoszenia projektu i założeń wstępnych



b) Odniesienie do założeń wstępnych

1. Co udało się zrealizować?

* Menu główne, wraz z wyborem poziomów.
* Podstawowe elementy gry (paletka, piłeczki, cegiełki)
* System fizyczny, uwzględniający zmiany prędkości piłeczki (cegiełki zmieniające przyspieszające i spowalniające), a także inne kąty odbicia niż początkowy (modyfikacja kąta odbicia ze względu na miejsce upadku piłeczki na paletkę oraz odbicia od narożników cegiełek).
* Inne rodzaje cegiełek, wraz z modyfikatorami rozgrywki (jest to w zasadzie to samo w obecnej implementacji)
* System dynamicznego ładowania poziomów z plików tekstowych. Można obejrzeć format poziomów w folderze *./Assets/Levels/*.

1. Czego nie udało się zrealizować?

* Eksport gry na Linux. Jest to zadanie wykonalne, choć czasochłonne, bo wymagałoby ono linkowania bibliotek na nowo, ale tym razem na Linuksie, co wiązałoby się także z koniecznością użycia maszyny wirtualnej.

6. Możliwość dalszej rozbudowy projektu

Projekt posiada dobrze uporządkowaną strukturę, co zdecydowanie sprzyja jego dalszej rozbudowie. Zwłaszcza zastosowanie dziedziczenia i polimorfizmu sprawia, że kod staje się wyjątkowo przejrzysty mimo swojej wielkości.

Projekt rozbudowywać można na wiele różnych sposobów. Możliwe pomysły to:

* Dodanie kolejnych rodzajów cegiełek o jeszcze innych właściwościach niż obecne.
* Przebudowa systemu poziomów w menu głównym tak, aby było możliwe podzielenie ich na segmenty tematyczne.
* Implementacja edytora poziomów oraz zewnętrzny serwer, służący do dzielenia się nimi z innymi graczami.
* Specjalne poziomy, odblokowywane tylko po spełnieniu określonych warunków np. zdobyciu ukrytej gwiazdki.
* Eksport gry na inne systemy operacyjne takie, jak Linux, a dla wytrwałych również MacOS, Android, czy iOS.
* Dodanie reklam, mikropłatności i przekształcenie gry w maszynkę do pieniędzy (niezalecane, jeśli celem jest utrzymanie dobrej opinii graczy).