Politechnika Świętokrzyska w Kielcach

Wydział Elektrotechniki, Automatyki i Informatyki
Katedra Informatyki, Elektroniki i Elektrotechniki

Programowanie obiektowe (Java) – projekt.

Temat projektu:

Tower Defense – gra.

Rok akademicki: 2021/22	Wykonali:
Semestr: LETNI	**Maja Hendzel** **Patrycja Kalita** **Marcin Kot**
Grupa dziekańska: 2ID12A	

1. Opis projektu.

W ramach projektu z przedmiotu „Programowanie obiektowe (Java)”, nasz zespół wykonał grę pod tytułem „Tower Defense”. Gra jest w języku angielskim.

Jest to gra strategiczna, czasu rzeczywistego. Tematyką gry są wieże wojenne, które niszczą nadlatujące, wrogie statki. Zadaniem gracza jest powstrzymywanie kolejnych fal wroga, za pomocą budowania wież. By wygrać należy przeżyć 10 map po 10 fal. Gracz wybiera wieżę, na którą go stać, niszczy wroga i zarabia. Za zabicie 30 wrogów, bez utraty życia, gracz otrzyma bonus pieniężny. Jeśli pokona wroga na mapie, to na kolejnej dostanie bonus pieniężny 100 * numer mapy. Każda mapa to więcej nadlatujących wrogów, i silniejszych od poprzenich. Każda broń ma inny zasięg, prędkość wystrzału, czy obrażenia.

Prędkość gry można zmienić używając lewej strzałki (zwolnienie) oraz prawej strzałki (przyspieszenie). Dodatkowo można edytować pierwszą mapę, dzięki czemu gracz może stworzyć swoją własną autorską mapę oraz zwiększyć lub zmniejszyć trudność początkowej fazy gry.

2. Informacje o projekcie.

Środowisko:

- IntelliJ IDEA 2021.1

Biblioteki:

- Slick2D,
- LWJGL (Lightweight Java Game Library)

3. Funkcjonalność projektu.

Po uruchomieniu gry widoczne są cztery przyciski: *PLAY*, *EDIT*, *INFO*, *QUIT*, są to również nasze funkcjonalności.

Pierwszy przycisk *PLAY*, uruchamia grę. Gra polega na kupowaniu wież, na które aktualnie gracza stać, i ustawianiu je w takich miejscach na mapie (zielone pola) by mogły zniszczyć wrogów zanim dotrą do końca mapy. Pomarańczowe pola to droga po której poruszają się wrogowie, nartomiast niebieskie pole to woda. Gra ma pomoc, wystarczy kliknąć przycisk F1. Rozgrywkę można zapauzować, przyśpieszyć, albo zwolnić. Za pomocą przycisku „ESC” wraca się do menu głównego.

Drugi przycisk *EDIT* daje możliwość edycji pierwszej mapy. Wystarczy wybrać jeden z rodzajów podłoża z menu po prawej stronie i można tworzyć swoją mapę. Za pomocą przycisku „ESC” wraca się do menu głównego.

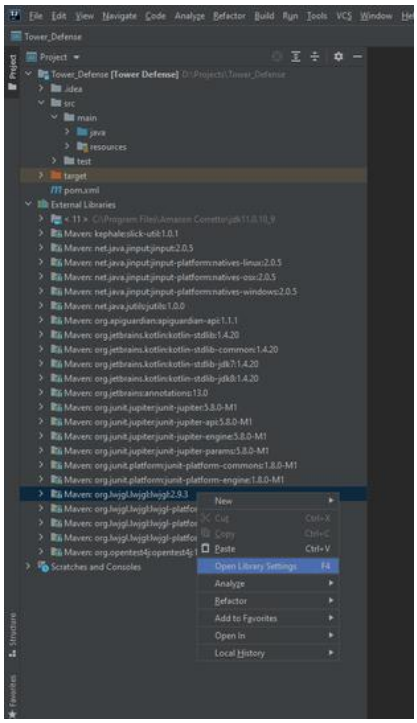
Kolejny przycisk *INFO* zawiera informacje o grze i o edycji pierwszej mapy.

Ostatni przycisk i zarazem funkcjonalność to *QUIT*. Jest to wyjście z gry.

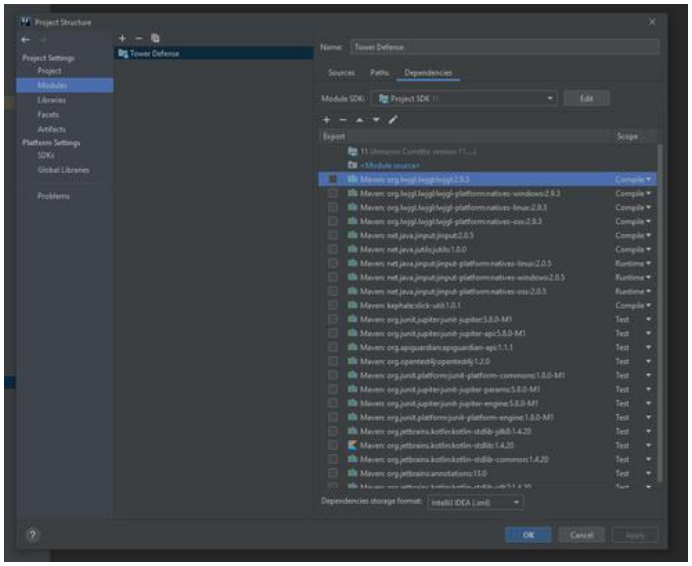
4. Sposób uruchomienia i obsługi projektu.

Grę można uruchamić za pomocą kompilacji klasy
..\Tower_Defense\src\main\java\data\Start.java w środowisku programistycznym.

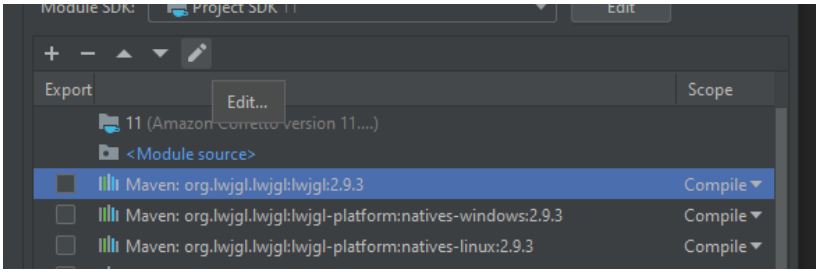
Dla środowiska IntelliJ przed skompilowaniem klasy *Start.java* należy kliknąć prawym przyciskiem myszy na plik „*lwjgl.jar*”. Następnie należy kliknąć opcję „Open Library Settings”.



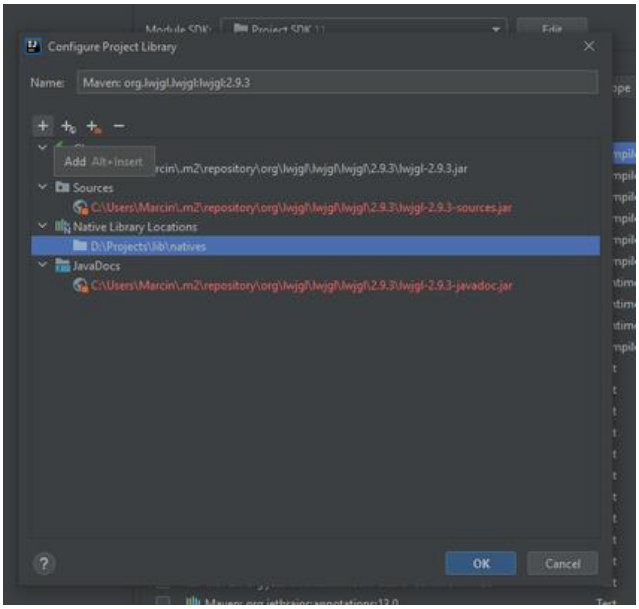
Następnie należy wybrać po lewej stronie zakładkę „Modules”.



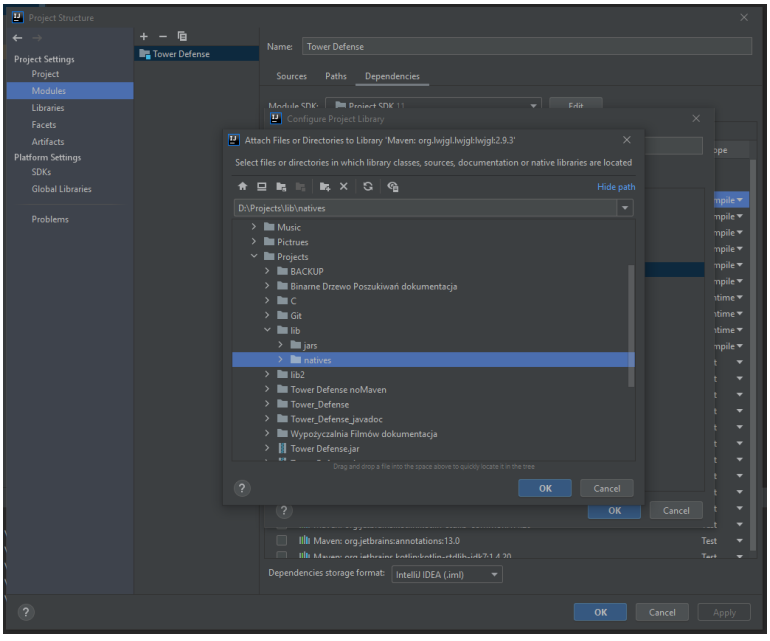
Należy wybrać z wyświetlonej listy „*lwjgl.jar*” i kliknąć przycisk „Edit” (symbol ołówka).



Natępnie należy kliknąć przycisk „Add” w nowym oknie.

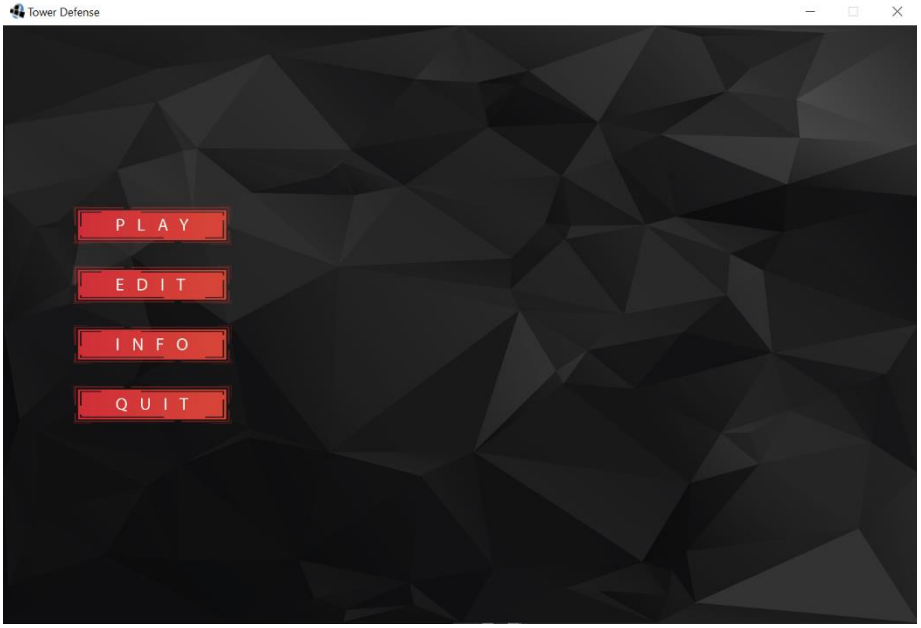


Ostatnim krokiem jest ustawienie lokalizacji folderu „native”.



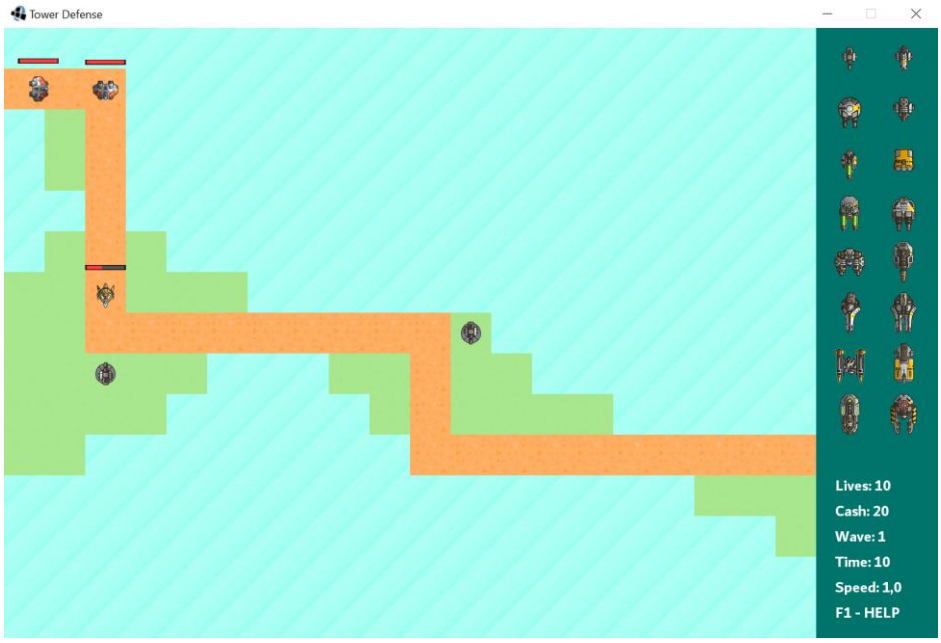
Przykład obsługi naszej gry:

Uruchomienie gry:

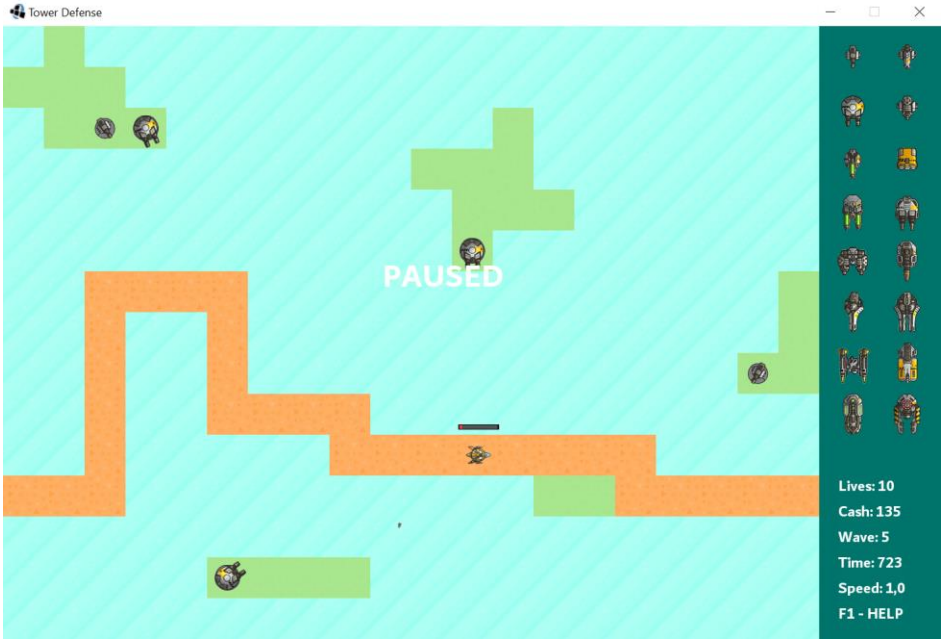


Wciśnięcie przycisku PLAY – uruchomienie gry:

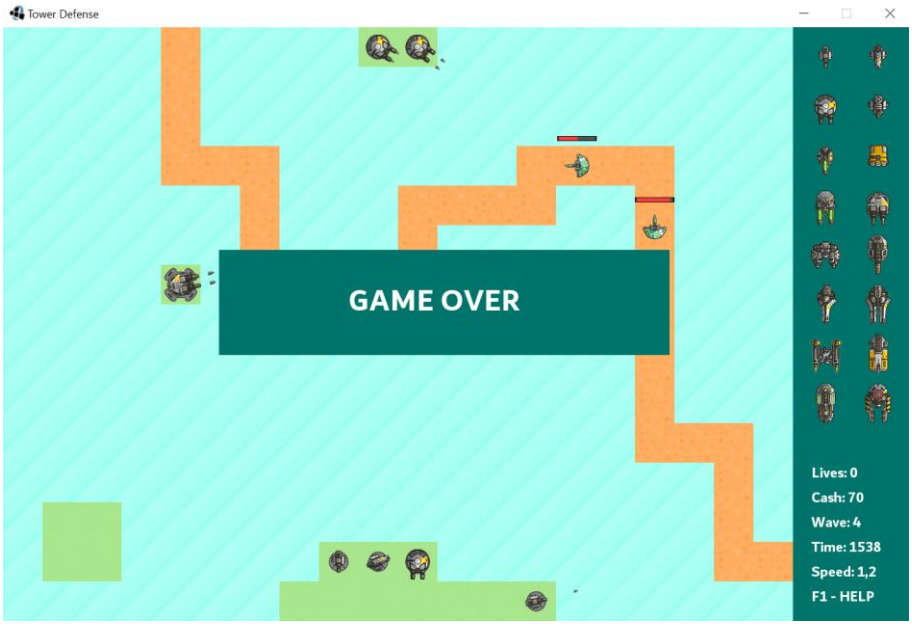
Pierwsza mapa, pierwsza fala:



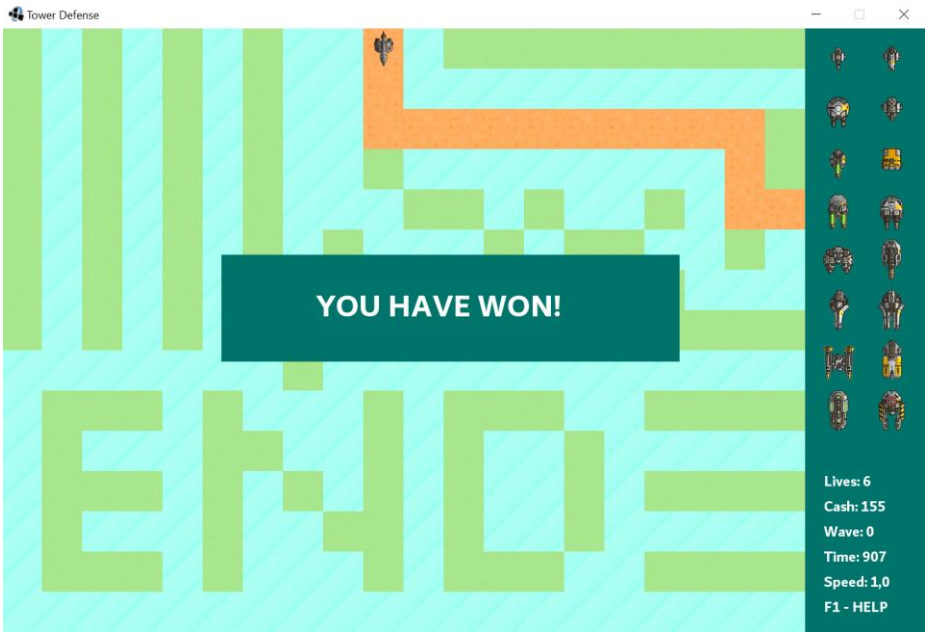
Zapauzowanie gry:



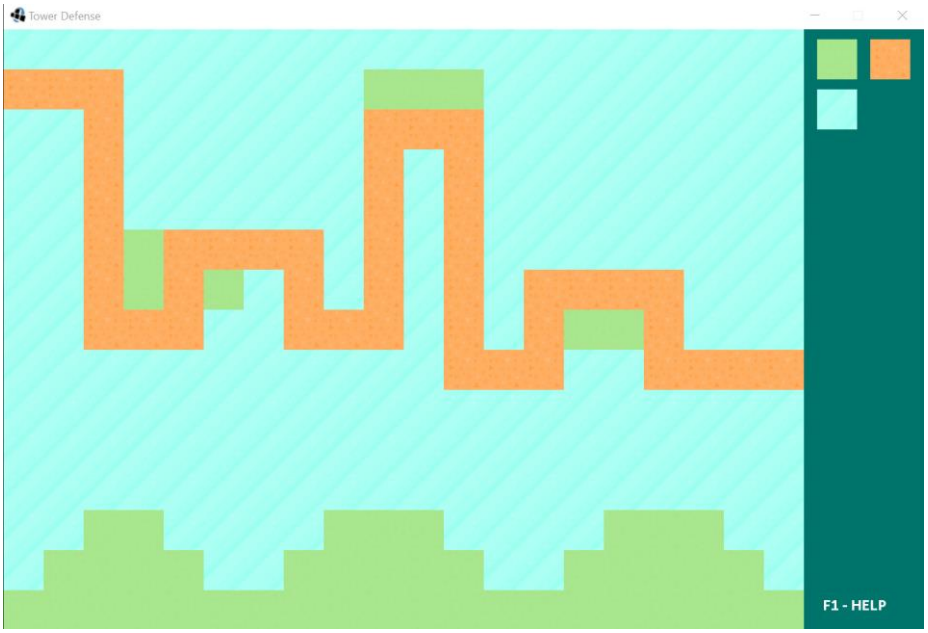
Przegrana gra:



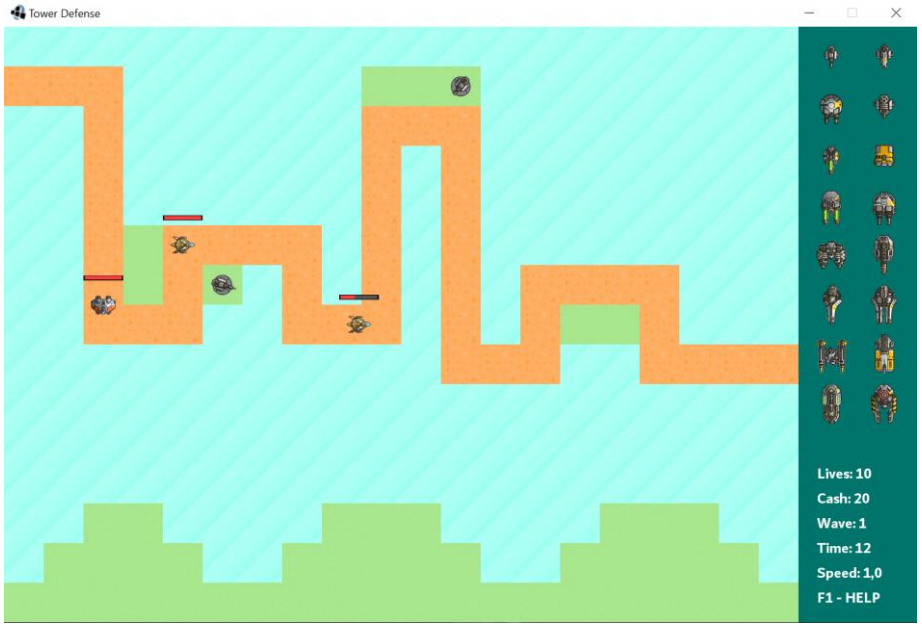
Wygrana gra:



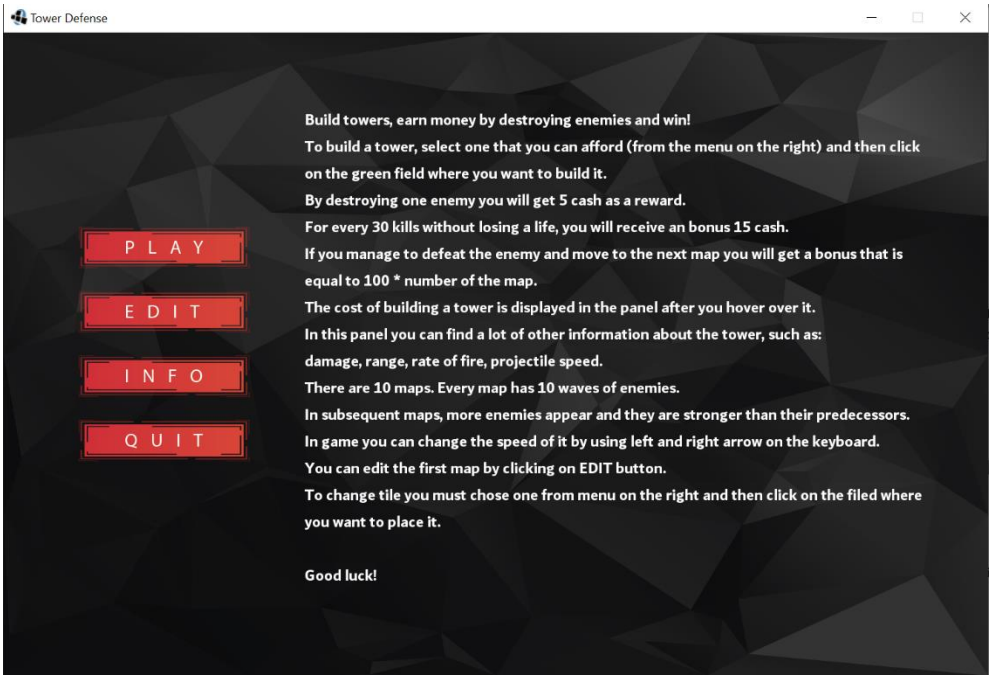
Wciśnięcie przycisku EDIT – edycja pierwszej mapy:



Wciśnięcie przycisku PLAY – włączenie gry, i przechodzenie rogrywki na naszej stworzonej mapie:



Wciśnięcie przycisko INFO – wyświetlenie informacji o grze i edycji pierwszej mapy:



5. Informacje na temat stworzonych klas, metod, funkcji z opisem podstawowej funkcjonalności oraz ich przeznaczeniem.

Stworzone klasy, metody i funkcje z folderu: ..\Tower_Defense\src\main\java**controllers**.

Plik **Graphic.java**:

Klasa Graphic m.in. ustawia tytuł okna „Tower Defense”, sprawdza czy wystąpiła kolizja dwóch obiektów, rysuje teksture czworokąta w podanych współrzędnych, zwraca obiekty klasy Texture, czy też oblicza kąt o jaki pocisk/rakieta musi się obrócić aby wycelować w przeciwnika.

Metoda StartSession()

- Ustawia tytuł okna "Tower Defense".
- Ustawia rozmiar wyświetlanego okna.
- Tworzy natywne okno.
- Zwraca wyjątek jeśli nie można ustawić trybu wyświetlania okna lub go stworzyć.
- Ustawia orthographic projection (czyli reprezentację trójwymiarowych obiektów w dwóch wymiarach) pomiędzy 0 i 600 na osi X, pomiędzy 400 i 0 na osi Y oraz pomiędzy 1 i -1 na osi Z określającej odległość od renderowanego obiektu.
- Umożliwia umieszczanie tekstur na ekranie.
- Pozwala na blendowanie tła na kanale alfa - czyli przezroczystość obrazu.
- Ustawienia dla kanału alfa i sposobu blendowania.

Metoda CheckCollision()

- Sprawdza czy wystąpiła kolizja dwóch obiektów.
- Metoda przyjmuje takie argumenty jak:
  - x1 - współrzędne pierwszego obiektu na osi x,
  - y1 - współrzędne pierwszego obiektu na osi y,
  - width1 - szerokość pierwszego obiektu,
  - height1 - wysokość pierwszego obiektu,
  - x2 - współrzędne drugiego obiektu na osi x,
  - y2 - współrzędne drugiego obiektu na osi y,
  - width2 - szerokość drugiego obiektu,
  - height2 - wysokość drugiego obiektu.
- Metoda zwraca true jeśli wystąpiła kolizja. W przeciwnym wypadku zwraca false.

Metoda DrawTexture()

- Rysuje tekstury czworokąta w podanych współrzędnych (będzie się tam znajdował jego lewy górny róg) o podanych rozmiarach i teksturze.
- Metoda przyjmuje takie argumenty jak:
  - texture - obiekt klasy Texture - wczytana tekstura z pliku,
  - x - pozycja na osi x, w której ma się znajdować lewy górny róg czworokąta,
  - y - pozycja na osi y, w której ma się znajdować lewy górny róg czworokąta,
  - width - szerokość prostokąta,
  - height - wysokość prostokąta.
- Powiązuje określony kontekst GL z teksturą.
- Translacja współrzędnych czworokąta względem okna na współrzędne poszczególnych wierzchołków względem siebie.
  * lewy górny: x = 0, y = 0 | prawy górny: x = 1, y = 0
  * lewy dolny: x = 0, y = 1 | prawy dolny: x = 1, y = 1
- Zapobieganie przed "rozrywaniem ekranu" - ang. screen tearing.

Metoda DrawRotatedTexture()

- Obsługuje tekstury, które mają się obracać wokół własnej osi.
- Rysuje tekstury czworokąta względem podanych współrzędnych o podanych rozmiarach i teksturze uwzględniając obrót wokół środka tego czworokąta.
- Metoda przyjmuje takie argumenty jak:
  - texture - obiekt klasy Texture - wczytana tekstura z pliku,
  - x - pozycja na osi x, w której ma się znajdować lewy górny róg czworokąta,
  - y - pozycja na osi y, w której ma się znajdować lewy górny róg czworokąta,
  - width - szerokość prostokąta,
  - height - wysokość prostokąta,
  - angle - kąt o jaki należy obrócić czworokąt.
- Powiązuje określony kontekst GL z teksturą.
- Translacja współrzędnych czworokąta względem okna na współrzędne poszczególnych wierzchołków względem siebie (wybranie środka czworokąta).
- Obrót czworokąta.
- Translacja współrzędnych czworokąta względem okna na współrzędne poszczególnych wierzchołków względem siebie (wybranie lewego górnego rogu czworokąta).
- Zapobieganie przed "rozrywaniem ekranu" - ang. screen tearing.

Metoda LoadTexture()

- Metoda zwraca obiekt klasy Texture - teksturę pobraną z pliku z podanej ścieżki o podanym typie pliku.
- IOException - obsługa wyjątku metody LoadTexture klasy Texture.
- Metoda przyjmuje takie argumenty jak:
  - path - ścieżka do pliku,
  - fileType - typ pliku.
- Metoda zwraca obiekt klasy Texture - wczytana tekstura z pliku.

Metoda FastLoad()

- Metoda automatyzująca część wprowadzania ścieżki i typu pliku wykorzystywanego jako tekstura. Dzięki niej wystarczy podać nazwę pliku który chcemy załadować jako teksturę. Zwraca obiekt klasy Texture.
- Metoda przyjmuje argument name - nazwa pliku.
- Metoda zwraca obiekt klasy Texture - wczytana tekstura z pliku.

Metoda CalculateAngle()

- Metoda obliczająca kąt o jaki pocisk/rakieta musi się obrócić aby wycelować w przeciwnika.
- atan2 to statyczna metoda obiektu Math, która oblicza kąt w radianach na podstawie współrzędnych (z obrotem w kierunku przeciwnym do ruchu wskazówek zegara).
- toDegrees konwertuje kąt w radianach na przybliżony kąt w stopniach.
- Metoda zwraca kąt w radianach o jaki należy obrócić teksturę.

Plik **Map.java**:

Klasa Map zawiera w sobie wczytywanie i zapisywanie map oraz funkcjonalności kafelków.

Metoda SaveMap()

- Metoda przegląda kolumnami wszystkie kafelki i zapisuje ich typy do zmiennej String. Szerokość i wysokość planszy w kafelkach zwraca metoda klasy TileGrid.
- Metoda przyjmuje takie argumenty jak:
  - mapName - nazwa pliku, w którym mają zostać zapisane dane o mapie.
  - grid - siatka mapy, która ma zostać zapisana w pliku.

Metoda LoadMap()

- Metoda wczytuje mapę z pliku konwertując jego zawartość.
- Metoda przyjmuje argument mapName - nazwa pliku zawierającego ciąg cyfr. Każda cyfra oznacza typ kafelka na mapie. Kolejność oznacza gdzie na mapie będzie się on znajdował.
- Metoda zwraca obiekt reprezentujący wczytaną mapę, na której program będzie pracował.
- Otwórz plik do odczytu.
- Zapisz dane z pliku do zmiennej data (pierwszą linię[w pierwszej linii jest cała zawartość pliku]).
- substring to metoda, która zwraca zmienną typu String będącą fragmentem innej zmiennej typu String, i * grid.getTilesHigh() + j to indeks znaku określającego typ aktualnie wczytywanego kafelka i oznacza kolumnę, grid.getTilesHigh() to rozmiar kolumny, j to wiersz w kolumnie na podstawie wczytanego konkretnego znaku jest ustawiany konkretny typ konkretnego kafelka na mapie.

Metoda GetTileType()

- Metoda zwracająca odpowiedni kafelek na podstawie znaku przekazanego przez parametr.
- Metoda przyjmuje argument ID - ID typu kafelka.
- Metoda zwraca typ kafelka.

Metoda GetTileID()

- Metoda zwracająca ID typu kafelka przekazanego jej przez parametr.
- Metoda przyjmuje argument tile - obiekt kafelek.
- Metoda zwraca znak reprezentujący typ kafelka.

Plik **StateManager.java**:

Klasa StateManager zawiera w sobie początkowy stan gry, czyli Główne Menu, obsługę stanu gry, Mierzenie liczby klatek na sekundę oraz zmianę stanu gry.

Metoda update()

- Obsługa stanu gry (menu, gra, edytor).
- Mierzenie liczby klatek na sekundę.
- Jeśli minęła sekunda to ustaw następną na 1000 ms po poprzedniej. Do zmiennej framesInLastSecond przypisz te zliczone w pętli w zmiennej framesInCurrentSecond. Wyzeruj liczbę klatek w aktualnej sekundzie.

Metoda setState()

- Metoda zmieniająca stan gry na ten przekazany przez parametr. Może to być na przykład MAINMENU, GAME, EDITOR.
- Metoda przyjmuje argument newState - stan gry na jaki ma zostać zmieniony aktualny.

Plik **Time.java**:

Klasa Time zawiera m.in. metodę która zwraca czas pomiędzy teraźniejszością, a ostatnią aktualizacją klatki wyświetlanego obrazu gry, metodę aktualizującą czas uruchomienia gry, czy statyczną metodę do zatrzymywania i wznawiania gry.

Metoda GetTime()

- Metoda zwraca typ long zwierający teraźniejszy czas.

Metoda GetTimeDelta()

- Metoda zwraca czas pomiędzy teraźniejszością a ostatnią aktualizacją klatki wyświetlanego obrazu gry.
- Jeśli czas od ostatniej wyświetlonej klatki jest większy niż 50 milisekund to zwróć 0,05.

Metoda GetDeltaFrameTime()

- Wartość statycznej zmiennej deltaFrameTime będącej atrybutem tej klasy zmienia się za każdym razem kiedy zostaje wyświetlona klatka obrazu. Wartość ta to czas pomiędzy poprzednią a aktualnie wyświetlaną klatką obrazu.
- Metoda zwraca 0 jeśli gra została zatrzymana, d * multiplier w przeciwnym przypadku.

Metoda Update()

- Aktualizuje czas uruchomienia gry.

Metoda ChangeGameSpeed()

- Statyczna metoda zmieniająca tempo gry.
- Metoda przyjmuje argument change - nowe tempo gry.

Metoda Pause()

- Statyczna metoda do zatrzymywania i wznawiania gry.

Metoda GetGameSpeed()

- Metoda zwraca tempo gry.

Metoda IsPaused()

- Metoda zwraca true lub false w zależności od tego czy gra jest zatrzymana.

Stworzone klasy, metody i funkcje z folderu: ..\Tower_Defense\src\main\java**data**.

Plik **Checkpoint.java:**

Klasa Checkpoint - Obiekt tej klasy reprezentuje kafelek na który przeciwnik będzie się przemieszczał.

Plik **Editor.java**:

Klasa Editor wczytuje pierwszą mapę, do jej edycji.

Konstruktor tej klasy wczytuje pierwszą mapę, indeks typu kafelka ustawia na 0. Pobiera typy kafelków. Wczytuje teksturę tła, które jest wyświetlane po prawej stronie ekranu od współrzędnej x = 1280. Tworzy interfejs użytkownika. Ustawia zmienną determinującą czy użytkownik chce aby pomoc była wyświetlana na false.

Metoda setupUI()

- Metoda tworząca interfejs użytkownika w Edytorze mapy.
- Utwórz obiekt klasy UI.
- Stwórz menu przycisków z jego lewym górnym rogiem na pozycji x = 1280, y = 16. O szerokości 192 px i wysokości 960 px.
- Pobierz i zapisz w atrybucie tilePickerMenu referencję do obiektu klasy menu będącego częścią kompozycji obiektu klasy UI.
- Dodaj przyciski odpowiadające za kafelki trawy, drogi oraz wody.

Metoda update()

- Rysowanie oraz obsługa myszy i klawiatury.

Metoda draw()

- Rysuje tło wyboru kafelków, mapę, menu kafelków oraz pomoc.

Metoda setTile()

- Metoda zmieniająca typ terenu kafelka, który znajduje się pod kursorem myszki.
- Przed próbą zmiany sprawdza czy kursor myszy znajduje się nad planszą.

Plik **Enemy.java**:

Klasa Enemy zawiera m.in. metodę która sprawdza czy przycisk dotarł do punktu kontrolnego, metodę która szuka kierunku w którym może się przemieszczać przeciwnik oraz taką metodę co zmienia stan wroga na „martwy".

Konstruktor przyjmuje takie argumenty jak:

- texture - tekstura typu przeciwnika
- startTile - pozycja startowa przeciwnika
- grid - kopia mapy gry
- width - szerokość
- height - wysokość
- speed - prędkość poruszania się
- Lista checkpoint przechowuje każdy zakręt na drodze przeciwnika
- Tablica directions odnosi się do osi x i y

Metoda update()

- Przesuwa wrogą jednostkę o iloczyn czasu jaki upłynął od ostatniej wyświetlonej klatki i prędkości tej jednostki w kierunku pobranym z listy punktów kontrolnych.
- Jeśli jest to pierwsze wywołanie tej metody zmienia ona tylko logiczną wartość zmiennej first na false.
- Jest to zrealizowane w ten sposób ponieważ po uruchomieniu gry, różnica w czasie (wartość zwrócona przez metodę getDelta() czyli czas teraźniejszy minus czas kiedy została dokonana ostatnia aktualizacją klatki wyświetlanego obrazu gry, ta druga składowa będzie miała wartość 0 ponieważ żadna klatka do tej pory nie została wyświetlona) będzie bardzo duża co sprawi że wrogie jednostki zostaną przesunięte poza okno gry i nie będą widoczne. * currentCheckpoint + 1 ponieważ w tym przypadku zmienna ta oznacza na którym checkpoincie aktualnie znajduje się przeciwnik a nie do którego zmierza. Więc musi sprawdzać czy od tego punktu, na którym jest ma dokąd podążać.
- Jeśli przeciwnik dotarł do ostatniego punktu swojej drogi to wywołaj metodę obsługującą takie zdarzenie.

Metoda getxEnemyDirection()

- Metoda zwraca 1 jeśli przeciwnik porusza się w prawo, -1 jeśli porusza się w lewo i 0 jeśli nie przemieszcza się na osi x.

Metoda getyEnemyDirection()

- Metoda zwraca 1 jeśli przeciwnik porusza się w dół, -1 jeśli porusza się do góry i 0 jeśli nie przemieszcza się na osi y.

Metoda endOfRoadReached()

- Jeśli przeciwnik dotrze do końca drogi zmniejsz liczbę żyć gracza o jeden, wyzeruj licznik zabitych przeciwników bez utraty życia i usuń przeciwnika.

Metoda checkpointReached()

- Sprawdza czy przeciwnik dotarł do punktu kontrolnego. Trzy piksele granicy błędu ponieważ nie zawsze w momencie sprawdzania przeciwnik będzie idealnie w miejscu, w którym powinien być. Jest to spowodowane tym że jego pozycja jest aktualizowana co klatkę obrazową o wartość równą czas jaki upłynął od poprzedniej wyświetlonej klatki obrazowej razy ewentualnie jego prędkość. Czasami może się zdarzyć większe opóźnienie i wtedy przeciwnik przemieści się o więcej pikseli niż zazwyczaj - stąd możliwe rozbieżności.

- Jeśli przeciwnik znajduje się w ustalonym przedziale pikseli odpowiadającym kafelkowi, który jest punktem kontrolnym to:
  - ustaw wartość zmiennej reached na true.
  - przypisz do aktualnej pozycji przeciwnika pozycję punktu kontrolnego.
- Metoda zwraca prawda lub fałsz w zależności od tego czy przeciwnik dotarł do punktu kontrolnego.

Metoda populateCheckpointList()

- Wypełnia listę checkpoints kafelkami na których przeciwnik musi zmienić kierunek poruszania się.
  Zostało to zrealizowane w następujący sposób:
- Do atrybutu obiektu directions przypisz kierunek, w którym może się przemieszczać przeciwnik.
- Do atrybutu obiektu checkpoints dodaj nowy punkt kontrolny - jest to kierunek oraz kafelek, do którego będzie podążał przeciwnik.
- Stwórz zmienną counter i zainicjuj ją wartością 0.

- Stwórz zmienna stuck typu boolean i zainicjuj ją wartością false - będzie ona służyła do określenia czy przeciwnik dotarł do miejsca, w którym nie może się poruszać dalej lub zmienił kierunek 20 razy.
- Dopóki przeciwnik nie utknął:
    - do tablicy typu int przypisz kierunek w którym będzie się poruszał przeciwnik.
    - Jeśli aktualny kierunek na osi X to 2 (czyli przeciwnik utknął) lub zmienił kierunek 20 razy to:
        - zawartość zmiennej stuck ustaw na true.
    - W przeciwnym przypadku:
        - do atrybutu obiektu directions przypisz kierunek, w którym może się przemieszczać przeciwnik z ostatniego punktu(kafelka) kontrolnego.
        - do atrybutu obiektu checkpoints dodaj nowy punkt kontrolny - jest to informacja o kierunku oraz kafelku, do którego przeciwnik musi dotrzeć a następnie zmienić kierunek poruszania się.
    - Zwiększ wartość zmiennej counter o 1. Zmienna ta jest wykorzystywana do wskazywania na ostatnio dodany punkt kontrolny na liście checkpoints.

Metoda findNextCheckpoint()

- Wyszukuje najbardziej oddalony kafelek tego samego typu co ten pod przeciwnikiem, znajdujący się w kierunku, w którym sie on przemieszczania.
- Jest to realizowane w następujący sposób:
- Dopóki zmienna found zawiera false:
    - Jeśli kafelek oddalony o tyle pól jaką wartość zawiera zmienna counter i w kierunku przemieszczania się przeciwnika jest innego typu terenu niz ten pod przeciwnikiem lub w sprawdzaniu wykraczasz poza mapę to:
        - wartość logiczną zmiennej found ustaw na true
        - zmniejsz wartość zmiennej counter o 1 tak żeby wskazywała na poprzedni kafelek.
        - przypisz do obiektu next, szukany kafelek
    - zwiększ wartość zmiennej counter
- Metoda przyjmuje takie argumenty jak
    - current - kafelek na którym aktualnie znajduje się przeciwnik
    - dir - kierunek, w którym przemieszcza się przeciwnik
- Metoda zwraca obiekt klasy Checkpoint zawierający informacje o tym, w którym kierunku i jak daleko ma się przemieszczać przeciwnik.

Metoda findNextDirection()

- Metoda szuka kierunku w którym może się przemieszczać przeciwnik.
- Realizuje to poprzez wczytanie informacji o sąsiednich czterech kafelkach.
- Jeśli typ terenu na którymś z nich jest taki sam jak ten pod przeciwnikiem i nie jest to kafelek, na którym przed chwilą był to będzie sie on przemieszczał w kierunku tego sąsiedniego kafelka. Kierunki mają priorytety po to aby przeciwnik był w stanie omijać przeszkody w sensowny sposób (poniżej priorytety kierunku wymienione malejąco):
  - Góra
  - Prawo
  - Dół
  - Lewo
- Jeśli typ terenu na sąsiednich czterech kafelkach nie jest taki sam jak ten pod przeciwnikiem to zatrzyma się on.
- Metoda przyjmuje argument current - aktualna pozycja przeciwnika.
- Metoda zwraca kierunek, w którym może się przemieszczać przeciwnik.

Metoda damage()

- Metoda, zmniejszająca liczbę reprezentującą zdrowie przeciwnika i zwiększająca liczbę zabitych przeciwników bez straty życia przez gracza.
- Metoda przyjmuje argument amount - wartość o jaką metoda zmniejszy liczbę reprezentującą zdrowie przeciwnika.

Metoda die()

- Metoda zmieniająca stan przeciwnika na "martwy"

Metoda draw()

- Rysuje przeciwnika na ekranie oraz jego pasek zdrowia.
- Grubość paska 8px, szerokość paska to szerokość kafelka czyli 64px. Położenie na osi y 16px nad kafelkiem przeciwnika. Szerokość tekstury oznaczającej w grze poziom zdrowia zmniejsza się proporcjonalnie względem stosunku posiadanego poziomu zdrowia do początkowego poziomu zdrowia.

Metoda reduceHiddenHealth()

- Metoda reduceHiddenHealth zmniejsza wartość ukrytego zdrowia przeciwnika o wartość przekazaną przez parametr. Używana po to aby nie wystrzeliwać kolejnych pocisków w przeciwnika, który zostanie zniszczony poprzez wystrzelone do tej pory pociski kierowane.
- Metoda przyjmuje argument amount - obrażenia zadawane przez ten typ pocisku.

Metoda restoreHiddenHealth()

- Metoda restoreHiddenHealth przywraca ukryte zdrowie przeciwnika o wartość przekazaną przez parametr. Używana kiedy na przykład pocisk kierowany trafi w przeciwnika, który zasłania cel.
- Metoda przyjmuje argument amount - obrażenia zadawane przez ten typ pocisku.

Plik **Game.java:**

Klasa Game zawiera m.in. dodawanie przycisków do interfejsu użytkownika, rysuje opis wieży reprezentującej przycisk menu, w miejscu gdzie znajduje się kursor myszy oraz metoda aktualizująca stan gry.

Konstruktor trybu gry. Przyjmuje takie argumenty jak:

- Wczytaj pierwszą mapę.
- Znajdź kafelek, na którym powinni się odradzać przeciwnicy.
- Wczytaj tło menu wyboru wież.
- Stwórz tablicę typów przeciwników.
- Wypełnij tablicę obiektami reprezentującymi typy przeciwników.
- Ustaw wartość reprezentującą możliwe do odrodzenia typy przeciwników poprzez wyciągnięcie wartości absolutnej z dzielenia liczby wszystkich typów wrogów przez numer mapy pomniejszony o 11. (w dziesiątej mapie będzie pełne spektrum możliwych typów przeciwników)
- Stwórz menadżera fal przeciwników. Przekaż do jego konstruktora tablicę typów wrogów, czas odradzania kolejnego przeciwnika w pierwszej fali, ilość przeciwników w pierwszej fali, liczbę reprezentującą spektrum możliwych do odrodzenia typów przeciwników.
- Utwórz obiekt klasy Player. Jako parametry przekaż pierwszą mapę gry oraz menadżera fal przeciwników.
- Ustaw początkowe wartości obiektu klasy Player reprezentujące ilość gotówki oraz poziom zdrowia.
- Stwórz przyciski menu wyboru wieży.
- Pobierz aktualny czas.

Metoda setupUI()

- Dodaje przyciski do interfejsu użytkownika.
- Tworzy menu przycisków.
- Pozycja pierwszego przycisku to 1280 na osi x oraz 16 na osi y.

Metoda updateUI()

- Rysuje interfejs użytkownika. Obsługuje wybieranie wież w menu.
- Rysuje opis wieży reprezentującej przycisk menu na którym znajduje się kursor myszy.
- Jeśli zarejestrowano nowe zdarzenie związane z myszą
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Light Cannon" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Light Warhead" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Twin Light Cannon" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Machine Gun" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Disruptor" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Light Antimatter Missile Launcher" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Twin Disruptors" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Twin Cannon" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Twin Machine Gun" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Heavy Cannon" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Heavy Warhead" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Twin Heavy Warhead" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Rocket Launcher" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Antimatter Missile Launcher" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "HellBore" w interfejsie użytkownika to podnieś ją.
    - Jeśli został naciśnięty LPM nad wieżą o nazwie "Pulson" w interfejsie użytkownika to podnieś ją.

Metoda drawTowerDescription()

- Rysuje opis wieży reprezentującej przycisk menu na którym znajduje się kursor myszy.
- Jeśli kursor myszy znajduje się na panelu menu wyboru wieży:
    - Przeglądaj przyciski wyboru wież:
        - Jeśli znajdziesz przycisk, na którym znajduje się kursor:
            - Przeglądaj typ wyliczeniowy typów wież:
                - - zapisz nazwę aktualnie przeglądanego typu wieży do zmiennej typu String o nazwie strThisTowerType.
                - Jeśli nazwa tekstury tego przycisku jest taka sama jak nazwa typu wyliczeniowego:
                  - narysuje tło.
                  - narysuj opis wieży na której znajduje się kursor.
                  - przerwij wykonywanie pętli.

Metoda update()

- Metoda aktualizująca stan gry.
- Jeśli obecny numer fali przeciwników to 10:
    - przejdź do kolejnej mapy.
- Rysuj mapę.
- Jeśli wszyscy przeciwnicy z tej fali są martwi to stwórz nową falę.
- Zaktualizuj listę przeciwników, wież. Obsłuż stawianie nowych wież.
- Narysuj tło dla menu wyboru wieży i statystyk.
- Zaktualizuj statystyki i menu wież.
- Jeśli gracz pokonał 30 przeciwników bez utraty zdrowia, przyznaj 15 gotówki jako bonus oraz wyświetl o tym informację.
- Jeśli mapa była zmieniana mniej niż 3 sekundy temu:
    - narysuj napis informujący o numerze aktualnej mapy.
- Jeśli gracz został pokonany:
    - Jeśli gra nie jest zatrzymana:
        - zatrzymaj grę.
        - ustaw czas wyjścia do głównego menu na za 5 sekund.
    - Narysuj tło.
    - Narysuj napis "GAME OVER".
    - Jeśli upłynął czas do wyjścia:
        - - wznów grę. (po to aby po ponownym wejściu do gry nie była ona zatrzymana)
        - - wyjdź do głównego menu.
- Jeśli gracz zwyciężył:
    - wywołaj metodę obsługującą to zdarzenie.

Metoda youHaveWon()

- Jeśli graczowi udało się obronić:
  - Jeśli gra nie jest zatrzymana:
    - zatrzymaj grę.
    - ustaw czas wyjścia do głównego menu na za 5 sekund.
  - Narysuj tło.
  - Narysuj napis "YOU HAVE WON!".
  - Jeśli upłynął czas do wyjścia:
    - wznów grę. (po to aby po ponownym wejściu do gry nie była ona zatrzymana)
    - wyjdź do głównego menu.

Metoda goToNextMap()

- Metoda obsługująca zmianę mapy na kolejną.
- Zeruje licznik fal. Usuwa wieże z mapy. Wczytuje nową mapę. Zapisuje czas kiedy została przełączona mapa.
- Wyszukuje miejsce, w którym będą się pojawiać przeciwnicy. Aktualizuje m.in trasy wszystkich typów przeciwników.
- Aktualizuje możliwe do odrodzenia typy przeciwników.
- Aktualizuje menadżera fal używając typów przeciwników ze zaktualizowanymi trasami.
- Aktualizuje gracza używając zaktualizowanej mapy oraz menadżera fal.

Metoda getWaveManager()

- Metoda zwraca statyczny atrybut tej klasy, którym jest menadżer fal przeciwników.

Metoda loadNextMap()

- Metoda wczytująca kolejną mapę.
- Metoda zwraca obiekt następnej mapy.

Metoda populateEnemyArray()

- Metoda wypełniająca tablicę typów przeciwników.

Metoda addEnemiesKilledWithoutHPLose()

- Statyczna metoda umożliwiająca zwiększanie liczby zabitych wrogów bez straty życia.
- Używana w klasie Enemy w przypadku gdy liczba reprezentująca zdrowie przeciwnika osiągnie wartość 0 lub mniejszą w skutek obrażeń odniesionych w wyniku kolizji z pociskiem.

Metoda resetEnemiesKilledWithoutHPLose()

- Statyczna metoda umożliwiająca resetowanie zmiennej zwierającej liczbę zabitych wrogów bez straty życia.
- Używana w klasie Enemy w przypadku gdy przeciwnikowi uda się dotrzeć do końca drogi.

Plik **MainMenu.java**:

Klasa MainMenu zawiera m.in. przyciski znajdujące się w menu głównym, metodę która odpowiada za włączenie stanów gry dostępnych w głównyum menu, informacje o grze oraz zmianę tekstur przycisków.

Konstruktor zawiera wczytywanie teksturę tła głównego menu gry. Utwórz interfejs użytkownika. Dodaje przyciski.

Metoda updateButtons()

- Metoda odpowiada za włączanie stanów gry dostępnych w głównym menu, wyświetlanie informacji o grze, oraz zmianę tekstur przycisków.
- Jeśli sprawdzono, że LPM jest wciśnięty po raz pierwszy od kąd nie był naciśnięty:
    - Jeśli kursor znajduje się nad przyciskiem "PLAY":
        - zmień stan gry na GAME - czyli wejdź do gry.
    - Jeśli kursor znajduje się nad przyciskiem "EDIT":
        - zmień stan gry na EDITOR - czyli wejdź do gry.
    - Jeśli kursor znajduje się nad przyciskiem "INFO":
        - zmień wartość logiczną zmiennej showINFO na przeciwną (sprawi to że w metodzie update zostanie wywołana metoda rysująca informacje o grze).
    - Jeśli kursor znajduje się nad przyciskiem "QUIT":
        - zamknij grę.
- W przeciwnym przypadku:
    - Wywołaj metodę changeButtonTextureDependingCursorPosition na rzecz wszystkich przycisków w menu. Metoda ta odpowiada za ustawianie odpowiedniej tekstury przycisków w zależności od aktualnego położenia kursora myszy.

Metoda update()

- Metoda rysująca tło menu, przyciski i wykorzystująca metodę updateButtons do obsługi przycisków i akcji z nimi związanych.

Metoda drawINFO()

- Metoda zawiera opis wyświetlanmy po kliknięciu przycisku INFO.

Plik **Player.java**:

Klasa Player zawiera m.in. do sprawdzenia czy gracz ma wystarczającą ilość pieniążków by mógł kupić daną wieżę obronną, metoda która zmienia liczbę żyć gracza oraz metoda obsługująca rysowanie oraz umieszczenie trzymanej wieży.

Pierwszy konstruktor zawiera takie argumenty jak:

- grid - obiekt klasy TileGrid. Przekazuje informacje o siatce kafelek.
- waveManager - menadżer fal przeciwników.

Konstruktor zapisuje siatkę kafelków. Pobiera typy kafelków. Zapisuje referencję do menadżera fal przeciwników. Tworzy listę wież, która będzie służyła do przechowywania wież zbudowanych przez gracza.

Drugi konstruktor jest używany podczas zmiany mapy aby zacząć ją z pustą listą wież, nową siatką, oraz nowym menadżerem fal przeciwników. Stara pozostaje ilość żyć i gotówki. Przyjmowany argument grid to atrybut tej klasy to obiekt klasy TileGrid. Przechowuje informacje o siatce kafelek.

Metoda setup()

- Służy do ustawiania początkowych wartości atrybutów statycznych klasy Player.

Metoda modifyCash()

- Metoda sprawdza czy gracz posiada wystarczającą ilość gotówki na zakup wieży. Jeśli tak od do aktualnej liczby gotówki odejmuje wartość wieży i zwraca true. Jeśli nie zwraca false.
- Metoda przyjmuje argument amount - liczba o jaka ma zostać zmniejszona aktualna liczba gotówki.

Metoda grantCash()

- Metoda dodająca graczowi gotówkę.
- Metoda przyjmuje argument amount - liczba o jaka ma zostać dodana do aktualnej liczby gotówki.

Metoda modifyLives()

- Modyfikuje liczbę żyć gracza.
- Netoda przyjmuje argument amount - liczba o jaką liczba żyć ma zostać zmodyfikowana.

Metoda update()

- Metoda obsługująca rysowanie oraz umieszczanie trzymanej wieży, aktualizację listy wież i przeciwników.
- Sprawdza czy gracz został pokonany przez przeważające siły wroga.
- Jeśli został naciśnięty prawy przycisk myszy porzuć umieszczanie trzymanej wieży.
- Przypisuje true jeśli lewy przycisk myszy jest nadal naciśnięty co uniemożliwia ponowne wejście w instrukcje warunkową powyżej i tym samym sprawia że jej zawartość nie jest wykonywana wielokrotnie podczas pojedynczego naciśnięcia lewego przycisku myszy lub false w przeciwnym przypadku.

Metoda placeTower()

- Metoda umieszczająca wieżę na kafelku, nad którym znajduje się kursor myszy jeżeli gracz aktualnie trzyma wybraną wieżę oraz posiada wystarczającą ilość gotówki.
- Po próbie umieszczenia zmień wartość holdingTower na false (co będzie oznaczało że gracz aktualnie nie trzyma żadnej wieży) oraz przypisz null do zmiennej tempTower, która jest tymczasowym pojemnikiem na wieżę, która została wybrana i jeśli zostaną spełnione odpowiednie warunki zostanie umieszczona.

Metoda pickTower()

- Metoda podnosząca wieżę z UI wykorzystywana w klasie Game

Metoda getMouseTile()

- Metoda zwracająca kafelek nad którym znajduje się kursor myszy.

Metoda removeTowers()

- Opróżnia listę zbudowanych wież czyli usuwa je z mapy.

Plik **Projectile.java**:

Klasa Projectile to abstrakcyjna klasa podobna do klasy Wave. Odpowiada za pociski.

Konstruktor posiada takie argumenty jak:

- type - typ pocisku
- target - obiekt klasy Enemy reprezentujący cel
- x - współrzędne na osi x w których ma znaleźć się lewy górny róg tekstury pocisku
- y - współrzędne na osi y w których ma znaleźć się lewy górny róg tekstury pocisku
- width - szerokość tekstury pocisku
- height - wysokość tekstury pocisku

Metoda calculateDirection()

- Metoda odpowiadająca za strzelanie w odpowiednim kierunku.
- Oblicza odległość od wieży do celu. Metoda abs służy do obliczania wartości bezwzględnej.
- Oblicza stosunek przemieszczenia pocisku w dwóch osiach aby osiągnąć zamierzony cel.
- Stosunek ten będzie reprezentował stosunek prędkości pocisku w osi x i y.

Metoda calculateDirectionWithPrediction()

- Metoda odpowiadająca za strzelanie w odpowiednim kierunku.
- Oblicza odległość od wieży do celu. Metoda abs służy do obliczania wartości bezwzględnej.
- Oblicza stosunek przemieszczenia pocisku w dwóch osiach aby osiągnąć zamierzony cel.
- Stosunek ten będzie reprezentował stosunek prędkości pocisku w osi x i y.
- Nadpisana wersja metody przewiduje pozycję celu w zależności od prędkości pocisku i celu.
- Jej zadaniem jest symulowanie celowania Close-In Weapons System takich jak Phalanx. Broń taka posiada dużą celność w przypadku kiedy cel porusza się ruchem jednostajnym prostoliniowym.
- Zasada działania algorytmu:
- Oblicz odległość do celu.
- Oblicz czas jaki potrzebuje pocisk żeby pokonać dystans od wieży do celu.
- Znając prędkość i kierunek ruchu przeciwnika oblicz pozycję na osi x oraz y, w której będzie się on znajdował po upłynięciu wcześniej obliczonego czasu.
- Dopóki dystans dla nowych współrzędnych jest różny od dystansu dla starych przewidywanych współrzędnych, w których pocisk miałby trafić w cel (prawie zawsze program potrzebuje dwóch iteracji):

        - do nowej odległości do celu przypisz dystans od wieży (tak na prawdę od miejsca w którym pojawi się pocisk w wieży) do celu.
        - do starej odległości do celu przypisz tą obliczoną w linii wyżej.
        - oblicz czas jaki potrzebuje pocisk żeby pokonać nową odległość od wieży do celu.
        - oblicz gdzie na osi x i y będzie się znajdował przeciwnik po pokonaniu przez pocisk nowego dystansu.
    - Zapisz przewidywane współrzędne, w których pocisk trafi w cel.
    - Zaktualizuj kąt natarcia pocisku.
    - Reszta metody jest taka sama jak w klasie abstrakcyjnej Projectile.

Metoda damage()

- Metoda zmniejsza ilość zdrowia trafionego przeciwnika i niszczy pocisk.
- Zmniejsz ilość zdrowia trafionego przeciwnika o wartość obrażeń jaką zadaje ten pocisk.
- Zniszcz pocisk.

Metoda update()

- Jeśli pocisk nie trafił jeszcze w cel lub uprzednio nie skończył się jego zasięg:
    - zapisz aktualną pozycję pocisku.
    - przemieść pocisk.
    - dodaj przebyty dystans w tej aktualizacji do całkowitego przebytego dystansu przez ten pocisk.
    - Jeśli całkowity przebyty dystans tego pocisku jest mniejszy lub równy jego zasięgowi:
        - zaktualilzuj listę przeciwników. [ze statycznego atrybutu o typie WaveManager klasy Game poprzez statyczną metodę getWaveManager() pobierana jest lista przeciwników w obecnej fali]
        - Iteruj po liście przeciwników:
            - Jeśli wykryjesz kolizję pocisku z przeciwnikiem z listy:
                - przypisz do atrybutu hittedEnemy przeciwnika z którym wystąpiła kolizja.
                - zadaj obrażenia i zniszcz pocisk.
                - przerwij wykonywanie się pętli for().
        - Wywołaj metodę rysującą.
    - Jeśli całkowity przebyty dystans tego pocisku jest większy niż jego zasięg:
        - zniszcz pocisk.

Metoda draw()

- Metoda rysująca pociski niekierowane.
- Rysuje teksturę pocisku obróconą o kąt który został obliczony w konstruktorze pocisku czyli podczas wystrzału.
- Później kąt ten się nie zmienia i jest zawarty w zmiennej angle (typ float).

Metoda drawGuidedMissile()

- Metoda rysująca pociski kierowane. Obraca teksturę pocisku w trakcie lotu tak aby była skierowana w cel.
- Jest to realizowane poprzez obliczanie kąta za każdym razem kiedy pocisk jest rysowany. Wynik tych obliczeń jest przekazywany jako czwarty parametr metody DrawQuadTextureRotate.
- Metoda ta jest używana w typie pocisku "Rocket" poprzez nadpisanie metody draw właśnie tą metodą.

Metoda calculateAngle()

- Metoda obliczająca kąt o jaki pocisk/rakieta musi się obrócić aby wycelować w przeciwnika.
- atan2 to statyczna metoda obiektu Math, która oblicza kąt w radianach na podstawie współrzędnych (z obrotem w kierunku przeciwnym do ruchu wskazówek zegara).
- toDegrees konwertuje kąt w radianach na przybliżony kąt w stopniach.

Metoda calculateDistance()

- Zwraca odległość od pocisku do przeciwnika.
- Metoda zawiera argument enemy - obiekt przeciwnika do którego dystans ma zostać obliczony.
- Metoda zwraca wartość jest przekątną prostokąta o bokach xDistance oraz yDistance czyli odległością od pocisku do przeciwnika.

Metoda getNewTarget()

- Metoda wyszukuje nowy, najbliższy oraz posiadający dodatnią liczbę reprezentującą jego zdrowie oraz ukryte zdrowie cel dla pocisku.
- Do zmiennej closest typu Enemy przypisz null.
- Do zmiennej closestDistance przypisz największą wartość jaką można zapisać w typie float.
- Przeglądając listę przeciwników znajdź takiego, który jest najbliżej oraz jego ukryte zdrowie jak i zdrowie jest większe od zera.

- Jeśli został znaleziony cel spełniający powyższe kryteria przypisz jego referencję do atrybutu target.

Metoda calculateDistanceAxis()

- Metoda obliczająca przekątną prostokąta. Jest używana do obliczania odległości między pociskiem a celem.
- Metoda zawiera następujące argumenty:
  - xTarget - pozycja celu na osi x
  - yTarget - pozycja celu na osi y
- Metoda zwraca odległość od pocisku do celu.

Metoda xTargetPositionAfter()

- Oblicza pozycję na osi x w jakiej będzie się znajdował cel po upływie czasu przekazanego jako parametr.
- Metoda zawiera argument time - czas po upływie, którego chcemy poznać pozycję przeciwnika.
- Metoda zwraca pozycje na osi x.

Metoda yTargetPositionAfter()

- Oblicza pozycję na osi y w jakiej będzie się znajdował cel po upływie czasu przekazanego jako parametr.
- Metoda zawiera argument time - czas po upływie, którego chcemy poznać pozycję przeciwnika.
- Metoda zwraca pozycje na osi y.

Plik **ProjectileType.java**:

Enum inaczej typ wyliczeniowy ProjectileType zawiera typy pocisków, i wartość jakich zadają obrażeń, ich prędkość oraz zasięg. 200 większy zasięg mają rakiety niż namierzania przez wyrzutnię, jest to zapas producenta po to aby rakiety częściej osiągały oddalający się cel lub cel zmieniony w trakcie lotu.

Plik **Start.java**:

Klasa Start importuje wszystko z biblioteki GL11 oraz wszystko z klasy Graphic. Konstruktor zawiera wywołanie statycznej metody klasy Graphic żeby zainicjować wywołania OpenGL. Zawiera również główną pętlę gry, która aktualizuje okno - wyświetl to co zostało narysowane. Odpytaj klawiaturę i myszkę. Ustawia określoną liczbę klatek na sekundę. Gra jest usypiana na dodatkowy dowolny czas w przypadku kiedy liczba FPS miałaby przekroczyć liczbę podaną w argumencie metody.

Plik **Tile.java**:

Klasa Tile zawiera m.in. metodę, która rysuje obiekt klasy Tile na ekranie, metoda która zwraca pozycję na osi x oraz metodę która zwraca pozycję na osi y.

Konstruktor zawiera takie argumenty jak:

- x - pozycja na osi x, w której ma znajdować się lewy górny róg czworokąta
- y - pozycja na osi y, w której ma znajdować się lewy górny róg czworokąta
- width - szerokość czworokąta
- height - wysokość czworokąta
- type - obiekt typu wyliczeniowego reprezentujący typ terenu

Metoda draw()

- Metoda rysująca obiekt klasy Tile na ekranie.
- Wykorzystuje do tego statyczną metodę DrawQuadTexture klasy Graphic.

Metoda getX()

- Metoda zwraca pozycję na osi x, w której znajduje się lewy górny róg czworokąta

Metoda getY()

- Metoda zwraca pozycję na osi y, w której znajduje się lewy górny róg czworokąta

Plik **TileGrid.java**:

Klasa TileGrid to klasa siatki kafelek.

Pierwszy konstruktor bez parametrów - wypełnia planszę takimi samymi kafelkami

Drugi konstruktor wypełniający plansze trzema rodzajami terenu. Argument tego konstruktora to newMap - dwuwymiarowa tabela, której elementy oznaczają jakim terenem i w którym miejscu na mapie ma być wypełniony odpowiadający temu polu kafelek. Do zmiennych prywatnych tileWide i tilesHigh przypisz szerokość oraz wysokość mapy w kafelkach.

Metoda setTile()

- Setter umożliwiający zmianę typu terenu kafelka
- Metoda zawiera takie argumenty jak:
  - xCoord - współrzędne na osi x kafelka do zmiany terenu
  - yCoord - współrzędne na osi y kafelka do zmiany terenu
  - type - nowy typ terenu

Metoda getTile()

Metoda zwraca obiekt klasy Tile (czyli kafelek) znajdujący się na współrzędnych przekazanych przez parametry metody o ile jest to pozycja na mapie. Jeśli wykracza ona poza mapę to metoda zwraca kafelek, który w metodzie FindNextCheckpoint klasy Enemy oznacza wykroczenie poza mapę podczas ustalania punktu kontrolnego. Współrzędne te są podane jako numery indeksów tablicy dwuwymiarowej składającej się z obiektów klasy Tile. Indeksy te pomnożone przez TILE_SIZE (64) to położenie kafelków na mapie w pikselach.

- Metoda zawiera takie argumenty jak:
  - xPlace - współrzędne na osi x kafelka
  - yPlace - współrzędne na osi y kafelka

Metoda draw()

- Metoda rysująca kafelki na planszy - są one obiektami klasy Tile.
- Wykorzystana jest tutaj metoda klasy Tile o nazwie Draw.

Plik **TileType.java**:

Enum TileType zawiera typ wyliczeniowy zawierający rodzaje terenu.

Konstruktor przyjmuje dwa argumenty:

- textureName - nazwa pliku
- buildable - możliwość budowania na tym terenie

Plik **Tower.java**:

Klasa abstrakcyjna zawiera m.in. metodę która wyszukuje najbliższy cel w zasięgu wieży, metodę która sprawdza czy przeciwnik jest w zasięgu wieży oraz metodę która rysuje wieżę nad wystrzeliwanymi pociskami.

Metoda acquireTarget()

- Metoda wyszukująca najbliższy cel w zasięgu wieży. Jeśli znajdzie cel zmienia wartość targeted na true.
- Metoda jesli znajdzie cel to zwraca jego obiekt, w przeciwnym przypadku zwraca null.

Metoda isInRange()

- Metoda sprawdza czy przeciwnik jest w zasięgu wieży.
- Metoda przyjmuje argument enemy - przeciwnik do którego dystans będzie sprawdzany.
- Metoda zwraca true jeśli przeciwnik jest w zasięgu, false w przeciwnym przypadku.

Metoda calculateDistance()

- Metoda zwraca odległość od wieży do przeciwnika. Zwracana wartość jest przekątną prostokąta o bokach xDistance oraz yDistance czyli odległością od wieży do przeciwnika.
- Metoda przyjmuje argument enemy - obiekt przeciwnika do którego dystans ma zostać obliczony.

Metoda calculateAngle()

- Metoda obliczająca kąt o jaki wieża musi się obrócić aby wycelować w przeciwnika.
- atan2 to statyczna metoda obiektu Math, która oblicza kąt w radianach na podstawie współrzędnych (z obrotem w kierunku przeciwnym do ruchu wskazówek zegara).
- toDegrees konwertuje kąt w radianach na przybliżony kąt w stopniach.

Metoda calculateAngleWithPrediction()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Zasada działania metody jest taka sama jak w klasie abstrakcyjnej Projectile. Dodatkowo metoda przyjmuje typ pocisku jako parametr. Zwraca kąt o jaki wieża musi się obrócić.

Metoda calculateDistanceAxis()

- Metoda obliczająca przekątną prostokąta. Jest używana do obliczania odległości między wieżą a celem.
- Metoda przyjmuje takie argumenty jak:
  - xTarget - pozycja celu na osi x
  - yTarget - pozycja celu na osi y
- Metoda zwraca odległość od wieży do celu.
- - TILE_SIZE ponieważ połowa rozmiaru wieży i połowa rozmiaru przeciwnika to rozmiar kafelka

Metoda xTargetPositionAfter()

- Oblicza pozycję na osi x w jakiej będzie się znajdował cel po upływie czasu przekazanego jako parametr.
- Metoda przyjmujwe argument time - czas po upływie, którego chcemy poznać pozycję przeciwnika
- Metoda zwraca pozycje na osi x

Metoda yTargetPositionAfter()

- Oblicza pozycję na osi y w jakiej będzie się znajdował cel po upływie czasu przekazanego jako parametr.
- Metoda przyjmuje argument time - czas po upływie, którego chcemy poznać pozycję przeciwnika
- Metoda zwraca pozycje na osi y

Metoda calculateProjectilePositionModifier()

- Metoda obliczająca o jaką wartość należy zmienić współrzędne względem wieży aby pocisk podczas wystrzału znajdował się w odpowiednim miejscu na wieży w zależności od kąta o jaki jest obrócona wieża.
- Metoda jest używana między innymi w wieży wyposażonej w rakiety po to aby dwa pociski zostały wystrzelone obok siebie, jednocześnie z miejsca w którym są widoczne na teksturze wieży, co daje efekt odpalenia pocisku z prowadnicy wyrzutni.
- Do obliczenia tych współrzędnych zostały użyte funkcje sinus, cosinus oraz kąt obrotu wieży w radianach.
- Metoda przyjmuje takie argumenty jak:
    - x - od tego parametru zależy o ile pikseli na osi x będzie oddalony pocisk od wieży
    - y - od tego parametru zależy o ile pikseli na osi y będzie oddalony pocisk od wieży

Metoda shoot()

- Abstrakcyjna metoda wykorzystywana do strzelania w klasach dziedziczących
- Metoda przyjmuje argument target - cel

Metoda updateEnemyList()

- Metoda aktualizująca listę przeciwników.
- Metoda przyjmuje argument newList - nowa lista przeciwników

Metoda stopTowerRotateAfterShotFor()

- Metoda obraca wieżę tylko jeśli minął zadany czas od osatniego wystrzału.
- Metoda przyjmuje argument stopRotateOfTowerTime

Metoda update()

- Metoda odpowiada za cykliczną obsługę działań wieży.
- Jeśli wieża nie jest wycelowana lub cel ma 0 lub mniej ukrytego zdrowia lub dystans do celu jest większy niż zasięg wieży:
    - znajdz nowy cel dla wieży.
- W przeciwnym przypadku:
    - sprawdź czy wieża nie powinna pozostać w bezruchu z uwagi na niedawny wystrzał.
    - Jeśli czas od ostatniego wystrzału jest większy lub równy czasowi co jaki wieża powinna strzelać:
        - wystrzel w kierunku przeciwnika, który jest celem.

    - ustaw czas od ostatniego wystrzału na 0.
- Jeśli referencja celu zawiera null lub cel jest martwy:
  przypisz false do zmiennej targeted reprezentującej wycelowanie (lub nie) wieży.
- Do czasu od ostatniego wystrzału dodaj czas jaki upłynął od ostatnio wyświetlonej klatki obrazu.
- Rysuj podstawę wieży pociski i wieżę.

Metoda draw()

- Metoda rysuje wieżę nad wystrzeliwanymi pociskami.
- Tekstury są rysowane w pętli w odpowiedniej kolejności (to znaczy: kolejne jej elementy nad poprzednimi)

Plik **TowerType.java**:

Enum inaczej typ wyliczeniowy TowerType zawiera typy wież obronych i m.in. ich zasięg, szybkość wystrzału, koszt.

Wieża może się składać z wielu tekstur, dlatego została użyta tablica tekstur.

Plik **Unit.java**:

Unit.java to interfejs. Służy do implementacji atrybutów i funkcji obsługujących pozycję, rozmiar oraz rysowanie.

Plik **Wave.java**:

Klasa Wave zawiera listę CopyOnWriteArrayList, ponieważ ten typ listy jest bezpieczny dla wątków. Operacje takie jak dodawanie, usuwanie są implementowane przez utworzenie nowej kopii podstawowej listy i nie jest ona modyfikowana jeśli istnieje jej iterator. Sam zaś iterator nie zgłasza ConcurrentModificationException oraz nie odzwierciedla dodań, usunięć ani zmian w liście od czasu utworzenia iteratora. Operacje zmiany elementów przy użyciu iteratorów nie są obsługiwane. Jest użyteczna kiedy występuje dużo więcej operacji odczytu niż modyfikacji na tej liście. W tym przypadku kiedy jeden wątek iteruje po liście a drugi(może to być też ten sam wątek) usuwa z niej przeciwnika ponieważ jest martwy, java.util zwraca wyjątek ConcurrentModificationException. Co oznacza że żaden wątek nie może modyfikować listy kiedy iterator po niej iteruje. Zmodyfikowana kopia listy zastępuje oryginalną tylko kiedy iteracja po liście oryginalnej się zakończy.

Konstruktor zawiera argumenty takie jak:

- spawnTime - odstęp czasu pomiędzy pojawianiem sie nowych przeciwników

    - enemyTypes - typ przeciwnika - obiekt klasy Enemy jest atrybutem klasy Wave (tej klasy), w nim są przechowywane informacje o tym jaki przeciwnik i gdzie ma zostać stworzony. Tworzy pierwszego przeciwnika.

Metoda update()

- Metoda ta przesuwa i rysuje na ekranie przeciwników oraz co ustalony czas tworzy nowych.
- Czas od ostatniego pojawienia się przeciwnika jest zwiększany o wartość zwróconą przez metodę Delta() należącą do klasy Time. Jest to realizowane co wyświetlenie się klatki obrazu. Tworzy nowych przeciwników dopóki ich liczba nie jest równa liczbie zadeklarowanych przeciwników na jedną falę. Sprawdza czy jacyś przeciwnicy w tej fali żyją. Jeśli tak to aktualizuje ich położenie i rysuje.
- Jeśli żaden z nich nie żyje i wszyscy przeciwnicy z tej fali zostali odrodzeni:
    - ustaw wartość zmiennej waveCompleted na true.

Metoda spawn()

- Dodaje nowego przeciwnika do listy. Typ przeciwnika jest losowany z zakresu zawartego w zmiennej enemySpawnTypeRange. Wartość tej zmiennej jest ustawiana w klasie Game podczas uruchamiania gry i rośnie przy każdej zmianie mapy tak aby zwiększać poziom trudności. Należy pamiętać, że zakres maksymalny zakres może być taki jak rozmiar tablicy - aktualnie 25. Przekazanie wartości 25 jako indeks tablicy dwudziestopięcio elementowej spowodowałoby wyrzucenie wyjątku IndexOutOfBoundException. Jako parametr metody nextInt jest przekazywana wartość 25 natomiast metoda ta zwraca liczbę z przedziału 0-24 czyli 25 różnych wartości co sprawia, że nie wykroczymy poza rozmiar tablicy.

Metoda getEnemyList()

- Getter zwracający listę przeciwników, którzy są w tej fali.

Plik **WaveManager.java**:

Klasa WaveManager zawiera m.in. metody które tworzą nowe fale, metodę która zwraca numer fali, czy też metodę która ustawia numer fali.

Metoda update()

- Jeśli przynajmniej część obecnej fali przeciwników żyje to wywołaj metodę aktualizującą przeciwników.
- W przeciwnym przypadku wywołaj nową falę.

Metoda newWave()

- Tworzy nowe fale i przypisuje najnowszy obiekt fali do referencji currentWave, która jest atrybutem tej klasy. Następnie inkrementuje licznik fal.
- Skraca czas między kolejnymi falami o 10%.
- Zwiększa ilość przeciwników o jeden.

Metoda getCurrentWave()

- Metoda zwraca obiekt reprezentujący ostatnio stworzoną falę.

Metoda getWaveNumber()

- Metoda zwraca numer fali.

Metoda setWaveNumber()

- Metoda ustawia numer fali.

Stworzone klasy, metody i funkcje z folderu: ..\Tower_Defense\src\main\java**enemies**.

Plik **AshdakAssaultShuttle.java**:

Klasa AshdakAssaultShuttle zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **AshdakBomber.java**:

Klasa AshdakBomber zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **AshdakHeavyFighter.java**:

Klasa AshdakHeavyFighter zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **AshdakInterceptor.java**:

Klasa AshdakInterceptor zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **AshdakSpecialAce.java**:

Klasa AshdakSpecialAce zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **GremakAssaultShuttle.java**:

Klasa GremakAssaultShuttle zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **GremakBomber.java**:

Klasa GremakBomber zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **GremakHeavyFighter.java**:

Klasa GremakHeavyFighter zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **GremakInterceptor.java**:

Klasa GremakInterceptor zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **HumanAssaultShuttle.java**:

Klasa HumanAssaultShuttle zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **HumanBomber.java**:

Klasa HumanBomber zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **HumanHeavyFighter.java**:

Klasa HumanHeavyFighter zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **HumanInterceptor.java**:

Klasa HumanInterceptor zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **OrthinAssaultShuttle.java**:

Klasa OrthinAssaultShuttle zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **OrthinBomber.java**:

Klasa OrthinBomber zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **OrthinHeavyFighter.java**:

Klasa OrthinHeavyFighter zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **OrthinInterceptor.java**:

Klasa OrthinInterceptor zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **PhidiAssaultShuttle.java**:

Klasa PhidiAssaultShuttle zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **PhidiBomber.java:**

Klasa PhidiBomber zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **PhidiHeavyFighter.java**:

Klasa PhidiHeavyFighter zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **PhidiInterceptor.java**:

Klasa PhidiInterceptor zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **YuralAssaultShuttle.java**:

Klasa YuralAssaultShuttle zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **YuralBomber.java**:

Klasa YuralBomber zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **YuralHeavyFighter.java**:

Klasa YuralHeavyFighter zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia.

Plik **YuralInterceptor.java**:

Klasa YuralInterceptor zawiera konstruktor który dla przeciwnika o tej nazwie ustawia m.in. wygląd, prędkość poruszania się i ilość zdrowia

Stworzone klasy, metody i funkcje z folderu: ..\Tower_Defense\src\main\java**projectiles**.

Plik **ProjectileAntimatterMissile.java**:

Klasa ProjectileAntimatterMissile zawiera w sobie m.in. w konstruktorze typ pocisu oraz cel i trzy nadpisane metody.

Metoda damage()

- Nadpisana metoda damage z abstrakcyjnej klasy Projectile.
- Jeśli trafiony przeciwnik nie jest pierwotnym celem (czyli pocisk trafił w przeciwnika, który zasłonił cel):
    - przywróć zdrowie przeciwnika
- Wywołaj metodę zadającą obrażenia trafionemu wrogowi i niszczącą pocisk.

Metoda draw()

- Zwykła metoda draw nadpisana metodą drawGuidedMissile(), która jest zdefiniowana w abstrakcyjnej klasie Projectile i obsługuje rysowanie pocisków kierowanych. Metoda draw jest wykorzystywana w metodzie update() więc takie nadpisanie powoduje zmianę działania oraz rezultatu metody update(), która również jest zdefiniowana w abstrakcyjnej klasie Projectile. Nadpisanie to jest wykonywane po to aby pocisk był obracany w stronę celu w takcie lotu.

Metoda update()

- Nadpisana metoda update() abstrakcyjnej klasy Projectile.
- Wykonaj to co zostało zdefiniowane w metodzie update() klasy Projectile.
- Wykonaj to co zostało zdefiniowane w metodzie calculateDirection() klasy Projectile.
- Przypisz aktualny czas lotu w milisekundch do atrybutu o nazwie projectileFilghtTime, który należy do tego obiektu.
- Jeśli cel posiada 0 lub mniej zdrowia lub jest martwy:
    - znajdź nowy cel dla tego pocisku.
    - Jeśli po wyszukiwaniu aktualny cel posiada 0 lub mniej zdrowia lub nie jest żywy:
        - zniszcz pocisk.
- Jeśli pocisk ma prędkość mniejszą niż 700 i czas lotu pocisku jest większy o co najmniej 50 w porównaniu z momentem ostatniego zwiększenia prędkości pocisku (celem drugiego warunku jest przyspieszanie pocisku co 50 ms):
    - zaktualizuj czas ostatniego zwiększenia prędkości pocisku.
    - przyspiesz pocisk o 10%.

Plik **ProjectileDisruptor.java**:

Klasa ProjectileDisruptor zawiera m.in. w konstruktorze typ pocisu oraz cel i jedną metodę.

Metoda damage()

- Nadpisana metoda damage z abstrakcyjnej klasy Projectile.
- Wywołaj metodę zadającą obrażenia trafionemu wrogowi i niszczącą pocisk.
- Spowolnij trafionego wroga o 15%.

Plik **ProjectileHeavyCannon.java**:

Klasa ProjectileHeavyCannon zawiera m.in. w konstruktorze typ pocisu oraz cel i jedną nadpisaną metodę.

Metoda calculateDirection()

- Celowanie z przewidywaniem położenia przeciwnika w momencie kiedy pocisk pokona dystans, który dzieli wieżę od celu.

Plik **ProjectileHeavyWarhead.java**:

Klasa ProjectileHeavyWarhead zawiera m.in. w konstruktorze typ pocisu oraz cel i jedną nadpisaną metodę.

Metoda calculateDirection()

- Celowanie z przewidywaniem położenia przeciwnika w momencie kiedy pocisk pokona dystans, który dzieli wieżę od celu.

Plik **ProjectileHellBore.java:**

Klasa ProjectileHellBore zawiera m.in. w konstruktorze typ pocisu oraz cel i dwie nadpisane metody.

Metoda calculateDirection()

- Celowanie z przewidywaniem położenia przeciwnika w momencie kiedy pocisk pokona dystans, który dzieli wieżę od celu.

Metoda damage()

- Nadpisana metoda damage z abstrakcyjnej klasy Projectile.
- Wywołaj metodę zadającą obrażenia trafionemu wrogowi i niszczącą pocisk.
- Spowolnij trafionego wroga o 35%.

Plik **ProjectileLightAntimatterMissile.java**:

Klasa ProjectileLightAntimatterMissile zawiera m.in. w konstruktorze typ pocisu oraz cel i trzy nadpisane metody.

Metoda damage()

- Nadpisana metoda damage z abstrakcyjnej klasy Projectile.
- Jeśli trafiony przeciwnik nie jest pierwotnym celem (czyli pocisk trafił w przeciwnika, który zasłonił cel):
  - przywróć zdrowie przeciwnika
- Wywołaj metodę zadającą obrażenia trafionemu wrogowi i niszczącą pocisk.

Metoda draw()

- Zwykła metoda draw nadpisana metodą drawGuidedMissile(), która jest zdefiniowana w abstrakcyjnej klasie Projectile i obsługuje rysowanie pocisków kierowanych. Metoda draw jest wykorzystywana w metodzie update() więc takie nadpisanie powoduje zmianę działania oraz rezultatu metody update(), która również jest zdefiniowana w abstrakcyjnej klasie Projectile. Nadpisanie to jest wykonywane po to aby pocisk był obracany w stronę celu w takcie lotu.

Metoda update()

- Nadpisana metoda update() abstrakcyjnej klasy Projectile.
- Wykonaj to co zostało zdefiniowane w metodzie update() klasy Projectile.
- Wykonaj to co zostało zdefiniowane w metodzie calculateDirection() klasy Projectile.
- Przypisz aktualny czas lotu w milisekundch do atrybutu o nazwie projectileFilghtTime, który należy do tego obiektu.
- Jeśli cel posiada 0 lub mniej zdrowia lub jest martwy:
  - znajdź nowy cel dla tego pocisku.
- Jeśli po wyszukiwaniu aktualny cel posiada 0 lub mniej zdrowia lub nie jest żywy:
  - zniszcz pocisk.
- Jeśli pocisk ma prędkość mniejszą niż 700 i czas lotu pocisku jest większy o co najmniej 50 w porównaniu z momentem ostatniego zwiększenia prędkości pocisku (celem drugiego warunku jest przyspieszanie pocisku co 50 ms):
  - zaktualizuj czas ostatniego zwiększenia prędkości pocisku.
  - przyspiesz pocisk o 10%.

Plik **ProjectileLightCannon.java:**

Klasa ProjectileLightCannon zawiera m.in. w konstruktorze typ pocisu oraz cel.

Plik **ProjectileLightWarhead.java:**

Klasa ProjectileLightWarhead zawiera m.in. w konstruktorze typ pocisu oraz cel.

Plik **ProjectileMachineGun.java**:

Klasa ProjectileMachineGun zawiera m.in. w konstruktorze typ pocisu oraz cel i jedną nadpisaną metodę.

Metoda calculateDirection()

- Celowanie z przewidywaniem położenia przeciwnika w momencie kiedy pocisk pokona dystans, który dzieli wieżę od celu.

Plik **ProjectilePulson.java**:

Klasa ProjectilePulson zawiera m.in. w konstruktorze typ pocisu oraz cel i jedną nadpisaną metodę.

Metoda calculateDirection()

- Celowanie z przewidywaniem położenia przeciwnika w momencie kiedy pocisk pokona dystans, który dzieli wieżę od celu.

Plik **ProjectileRocket.java**:

Klasa ProjectileRocket zawiera m.in. w konstruktorze typ pocisu oraz cel i trzy nadpisane metody.

Metoda damage()

- Nadpisana metoda damage z abstrakcyjnej klasy Projectile.
- Jeśli trafiony przeciwnik nie jest pierwotnym celem (czyli pocisk trafił w przeciwnika, który zasłonił cel):
  - przywróć zdrowie przeciwnika
- Wywołaj metodę zadającą obrażenia trafionemu wrogowi i niszczącą pocisk.
- Spowolnij trafionego wroga o 25%.

Metoda draw()

- Zwykła metoda draw nadpisana metodą drawGuidedMissile(), która jest zdefiniowana w abstrakcyjnej klasie Projectile i obsługuje rysowanie pocisków kierowanych. Metoda draw jest wykorzystywana w metodzie update() więc takie nadpisanie powoduje zmianę działania oraz rezultatu metody update(), która również jest zdefiniowana w abstrakcyjnej klasie Projectile. Nadpisanie to jest wykonywane po to aby pocisk był obracany w stronę celu w takcie lotu.

Metoda update()

- Nadpisana metoda update() abstrakcyjnej klasy Projectile.
- Wykonaj to co zostało zdefiniowane w metodzie update() klasy Projectile.
- Wykonaj to co zostało zdefiniowane w metodzie calculateDirection() klasy Projectile.
- Przypisz aktualny czas lotu w milisekundch do atrybutu o nazwie projectileFilghtTime, który należy do tego obiektu.
- Jeśli cel posiada 0 lub mniej zdrowia lub jest martwy:
    - znajdź nowy cel dla tego pocisku.
    - Jeśli po wyszukiwaniu aktualny cel posiada 0 lub mniej zdrowia lub nie jest żywy:
        - zniszcz pocisk.
- Jeśli pocisk ma prędkość mniejszą niż 700 i czas lotu pocisku jest większy o co najmniej 50 w porównaniu z momentem ostatniego zwiększenia prędkości pocisku (celem drugiego warunku jest przyspieszanie pocisku co 50 ms):
    - zaktualizuj czas ostatniego zwiększenia prędkości pocisku.
    - przyspiesz pocisk o 10%.

Plik **ProjectileTwinCannon.java:**

Klasa ProjectileTwinCannon zawiera m.in. w konstruktorze typ pocisu oraz cel i jedną nadpisaną metodę.

Metoda calculateDirection()

- Celowanie z przewidywaniem położenia przeciwnika w momencie kiedy pocisk pokona dystans, który dzieli wieżę od celu.

Stworzone klasy, metody i funkcje z folderu: ..\Tower_Defense\src\main\java**towers**.

Plik **AntimatterMissileLauncher.java**:

Klasa AntimatterMissileLauncher zawiera m.in. metodę która implementuje abstrakcyjną metodę shootabstrakcyjnej klasy Tower oraz dwie nadpisane metody.

Metoda shoot()

- Implementacja abstrakcyjnej metody shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje rakiety z trzech różnych prowadnic w zależności, od tego która z kolei jest to rakieta.
- Dzięki temu uzyskuje się efekt wystrzeliwania rakiet z prowadnic kolejno 0, 1, 2, 0, 1, 2, 0, 1, 2... itd.
- Po wystrzale rakiety z odpowiedniej prowadnicy jest redukowany poziom zdrowia przeciwnika, który jest celem.

Metoda draw()

- Nadpisana metoda draw abstrakcyjnej klasy Tower.
- Rysuje podstawę wieży, nad nią rakiety i nad rakietami wyrzutnię.

Metoda update()

- Nadpisana metoda update() abstrakcyjnej klasy Tower.
- Jeśli wieża nie jest wycelowana lub cel ma 0 lub mniej ukrytego zdrowia lub dystans do celu jest większy niż zasięg wieży:
  znajdz nowy cel dla wieży.
- W przeciwnym przypadku:
  - sprawdź czy wieża nie powinna pozostać w bezruchu z uwagi na niedawny wystrzał. [nie dotyczy tej wieży | czas stopu ustawiony na 0]
  - Jeśli czas od ostatniego wystrzału jest większy lub równy czasowi co jaki wieża powinna strzelać lub jeśli wieża jest w trakcie wystrzeliwania serii i jednocześnie czas od odpalenia ostatniej rakiety jest większy niż 0.038 sekundy:
    - ustaw czas od ostatnio wystrzelonej serii na 0.
    - wystrzel w kierunku przeciwnika, który jest celem.
    - zinkrementuj o jeden zmienną determinującą prowadnicę z której należy odpalić następną rakietę.
    - Jeśli wystrzelono rakiety z prowadnic 0, 1, i 2:
      - ustaw prowadnicę 0 jako następną, z której będzie odpalona rakieta.
      - zwiększ numer serii. [Cztery serie przed ponownym załadowaniem prowadnic. Łącznie 12 prowadnic i rakiet]
      - Jeśli została wystrzelona ostatnia seria:

        - ustaw numer serii na 0 co spowoduje że wieża będzie czekać na załadowanie prowadnic ("przełądowanie").
    - Jeśli aktualny numer serii oraz numer rakiety to 0:
        - ustaw czas od ostatniego wystrzału na 0;
- Jeśli referencja celu zawiera null lub cel jest martwy:
    - przypisz false do zmiennej targeted reprezentującej wycelowanie (lub nie) wieży.
- Utwórz zmienna temp i przypisz do niej czas od ostatnio odpalonej pojedynczej rakiety.
- Do czasu od ostatnio odpalonej pojedynczej rakiety dodaj czas jaki upłynął od ostatnio wyświetlonej klatki obrazu.
- Do zmiennej przechowującej czas od ostatnio wystrzelonych wszystki rakiet z 12 prowadnic dodaj czas jaki upłunął od ostatnio wyświetlonej klatki obrazu wykorzystując do tego zmienną temp oraz timeSinceLastLaunch.
- Rysuj wieżę i pociski w odpowiedniej kolejności używając metody draw().

Plik **Disruptor.java**:

Klasa Disruptor zawiera m.in. w konstruktorze typ wieży i jedną nadpisaną metodę.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy rozpraszacza.

Plik **HeavyCannon.java**:

Klasa HeavyCannon zawiera m.in. w konstruktorze typ wieży i dwie nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy ciężkiego działa.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Plik **HeavyWarhead.java**:

Klasa HeavyWarhead zawiera m.in. w konstruktorze typ wieży i dwie nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Plik **HellBore.java**:

Klasa HellBore zawiera m.in. w konstruktorze typ wieży i dwie nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy.
- Jeśli prędkość poruszania się celu jest mniejsza niż 10:
    - wyceluj w przeciwnika, który znajduje się najbliżej wieży.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Plik **LightAntimatterMissileLauncher.java**:

Klasa LightAntimatterMissileLauncher zawiera m.in. metodę która implementuje abstrakcyjną metodę shootabstrakcyjnej klasy Tower oraz dwie nadpisane metody.

Metoda shoot()

- Implementacja abstrakcyjnej metody shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje rakiety z trzech różnych prowadnic w zależności, od tego która z kolei jest to rakieta.
- Dzięki temu uzyskuje się efekt wystrzeliwania rakiet z prowadnic kolejno 0, 1, 2, 0, 1, 2, 0, 1, 2... itd.
- Po wystrzale rakiety z odpowiedniej prowadnicy jest redukowany poziom zdrowia przeciwnika, który jest celem.

Metoda draw()

- Nadpisana metoda draw abstrakcyjnej klasy Tower.
- Rysuje podstawę wieży, nad nią rakiety i nad rakietami wyrzutnię.

Metoda update()

- Nadpisana metoda update() abstrakcyjnej klasy Tower.
- Jeśli wieża nie jest wycelowana lub cel ma 0 lub mniej ukrytego zdrowia lub dystans do celu jest większy niż zasięg wieży:
  znajdz nowy cel dla wieży.
- W przeciwnym przypadku:
  - sprawdź czy wieża nie powinna pozostać w bezruchu z uwagi na niedawny wystrzał. [nie dotyczy tej wieży | czas stopu ustawiony na 0]
  - Jeśli czas od ostatniego wystrzału jest większy lub równy czasowi co jaki wieża powinna strzelać lub jeśli wieża jest w trakcie wystrzeliwania serii i jednocześnie czas od odpalenia ostatniej rakiety jest większy niż 0.038 sekundy:
    - ustaw czas od ostatnio wystrzelonej serii na 0.
    - wystrzel w kierunku przeciwnika, który jest celem.
    - zinkrementuj o jeden zmienną determinującą prowadnicę z której należy odpalić następną rakietę.
    - Jeśli wystrzelono rakiety z prowadnic 0, 1, i 2:
      - ustaw prowadnicę 0 jako następną, z której będzie odpalona rakieta.
      - zwiększ numer serii. [Cztery serie przed ponownym załadowaniem prowadnic. Łącznie 12 prowadnic i rakiet]
      - Jeśli została wystrzelona ostatnia seria:

        - ustaw numer serii na 0 co spowoduje że wieża będzie czekać na załadowanie prowadnic ("przełądowanie").
    - Jeśli aktualny numer serii oraz numer rakiety to 0:
        - ustaw czas od ostatniego wystrzału na 0;
- Jeśli referencja celu zawiera null lub cel jest martwy:
    - przypisz false do zmiennej targeted reprezentującej wycelowanie (lub nie) wieży.
- Utwórz zmienna temp i przypisz do niej czas od ostatnio odpalonej pojedynczej rakiety.
- Do czasu od ostatnio odpalonej pojedynczej rakiety dodaj czas jaki upłynął od ostatnio wyświetlonej klatki obrazu.
- Do zmiennej przechowującej czas od ostatnio wystrzelonych wszystki rakiet z 12 prowadnic dodaj czas jaki upłunął od ostatnio wyświetlonej klatki obrazu wykorzystując do tego zmienną temp oraz timeSinceLastLaunch.
- Rysuj wieżę i pociski w odpowiedniej kolejności używając metody draw().

Plik **LightCannon.java**:

Klasa LightCannon zawiera m.in. w konstruktorze typ wieży i jedną nadpisaną metodę.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy lekkiego działa.

Plik **LightWarhead.java**:

Klasa LightWarhead zawiera m.in. w konstruktorze typ wieży i jedną nadpisaną metodę.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy.

Plik **MachineGun.java**:

Klasa MachineGun zawiera m.in. w konstruktorze typ wieży i trzy nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy karabinu maszynowego.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Metoda update()

- Metoda odpowiada za cykliczną obsługę działań wieży.
- Jeśli wieża nie jest wycelowana lub cel ma 0 lub mniej ukrytego zdrowia lub dystans do celu jest większy niż zasięg wieży:
    - znajdz nowy cel dla wieży.
- W przeciwnym przypadku:
    - sprawdź czy wieża nie powinna pozostać w bezruchu z uwagi na niedawny wystrzał.
    - Jeśli czas od ostatniego wystrzału jest większy lub równy czasowi co jaki wieża powinna strzelać i magazynek nie jest pusty:
        - wystrzel w kierunku przeciwnika, który jest celem.
        - ustaw czas od ostatniego wystrzału na 0.
        - zmniejsz ilość amunicji w magazynku o jeden.
- Jeśli referencja celu zawiera null lub cel jest martwy:
    - przypisz false do zmiennej targeted reprezentującej wycelowanie (lub nie) wieży.
- Do czasu od ostatniego wystrzału dodaj czas jaki upłynął od ostatnio wyświetlonej klatki obrazu.
- Jeśli czas od ostatniego wystrzału jest większy niż 4 sekundy uzupełnij zawartość magazynka.
- Rysuj podstawę wieży pociski i wieżę.

Plik **Pulson.java**:

Klasa Pulson zawiera m.in. w konstruktorze typ wieży i dwie nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy.
- Po każdym oddanym strzale wyceluj w przeciwnika, który znajduje się najbliżej wieży.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Plik **RocketLauncher.java**:

Klasa RocketLauncher zawiera m.in. w konstruktorze typ wieży i jedną nadpisaną metodę.

Metoda shoot()

- Implementacja abstrakcyjnej metody shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje dwie rakiety z dwóch różnych prowadnic na raz.

Metoda draw()

- Nadpisana metoda draw abstrakcyjnej klasy Tower.
- Rysuje podstawę wieży a następnie jeśli czas do kolejnego wystrzału jest krótszy niż jedna sekunda to rysuje teksturę wyrzutni rakiet z załadowanymi rakietami. W przeciwnym przypadku rysuje teksturę pustej wyrzutni rakiet.
- Na koniec aktualizuje wystrzelone, niezniszczone rakiety.

Plik **TwinCannon.java**:

Klasa TwinCannon zawiera m.in. w konstruktorze typ wieży i dwie nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pociski z końców luf bliźniaczych dział.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Plik **TwinDisruptors.java:**

Klasa TwinDisruptors zawiera m.in. w konstruktorze typ wieży i jedną nadpisaną metodę.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pociski z końca luf bliźniaczych rozpraszaczy.

Plik **TwinHeavyWarhead.java**:

Klasa TwinHeavyWarhead zawiera m.in. w konstruktorze typ wieży i dwie nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pocisk z końca lufy.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Plik **TwinLightCannon.java**:

Klasa TwinLightCannon zawiera m.in. w konstruktorze typ wieży i jedną nadpisaną metodę.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pociski z końców luf bliźniaczych lekkich dział.

Plik **TwinMachineGun.java**:

Klasa TwinMachineGun zawiera m.in. w konstruktorze typ wieży i trzy nadpisane metody.

Metoda shoot()

- Nadpisana metoda shoot abstrakcyjnej klasy Tower.
- Wystrzeliwuje pociski z końców luf bliźniaczych karabinów maszynowych.

Metoda calculateAngle()

- Metoda odpowiadająca za celowanie przed przeciwnika z wyprzedzeniem, dzięki któremu wieża zyskuje 100% celności w przypadku kiedy przeciwnik porusza się ruchem jednostajnym prostoliniowym.
- Wywoływana metoda przyjmuje typ pocisku jako parametr żeby poznać jego prędkość. Zwraca kąt o jaki wieża musi się obrócić.

Metoda update()

- Metoda odpowiada za cykliczną obsługę działań wieży.
- Jeśli wieża nie jest wycelowana lub cel ma 0 lub mniej ukrytego zdrowia lub dystans do celu jest większy niż zasięg wieży:
    - znajdz nowy cel dla wieży.
- W przeciwnym przypadku:
    - sprawdź czy wieża nie powinna pozostać w bezruchu z uwagi na niedawny wystrzał.
    - Jeśli czas od ostatniego wystrzału jest większy lub równy czasowi co jaki wieża powinna strzelać i magazynek nie jest pusty:
        - wystrzel w kierunku przeciwnika, który jest celem.
        - ustaw czas od ostatniego wystrzału na 0.
        - zmniejsz ilość amunicji w magazynku o jeden.
- Jeśli referencja celu zawiera null lub cel jest martwy:
    - przypisz false do zmiennej targeted reprezentującej wycelowanie (lub nie) wieży.

- Do czasu od ostatniego wystrzału dodaj czas jaki upłynął od ostatnio wyświetlonej klatki obrazu.
- Jeśli czas od ostatniego wystrzału jest większy niż 4 sekundy uzupełnij zawartość magazynka.
- Rysuj podstawę wieży pociski i wieżę.

Stworzone klasy, metody i funkcje z folderu: ..\Tower_Defense\src\main\java**user_interface**.

Plik **Button.java**:

Klasa Button to klasa która odpowiada za przyciski.

Pierwszy jak i drugi konstruktor pobierają szerokość i wysokość z pliku graficznego.

Plik **UI.java**:

Klasa UI zawiera m.in. metodę rysującą tekst o podanych współrzędnych, metodę rysującąś przyciski z listy buttonList w oknie gry oraz metodę która upraszcza dodawanie nowego przycisku do siatki przyciswkó menu.

Metoda drawStringDescription()

- Metoda rysująca opis tekstowy w podanych współrzędnych (wszystkie te informacje są przekazywane przez parametr)
- Używana do wyświetlania informacji o aktualnym numerze mapy.
- Metoda zawiera takie argumenty jak:
  - x - położenie napisu na osi x
  - y - położenie napisu na osi y
  - text - tekst do wyświetlenia

Metoda drawStringHeading()

- Metoda rysująca nagłówek tekstowy w podanych współrzędnych (wszystkie te informacje są przekazywane przez parametr)
- Używana do wyświetlania informacji o aktualnym numerze mapy.
- Metoda zawiera takie argumenty jak:
  - x - położenie napisu na osi x
  - y - położenie napisu na osi y
  - text - tekst do wyświetlenia

Metoda drawString()

- Metoda rysująca tekst w podanych współrzędnych (wszystkie te informacje są przekazywane przez parametr)
- Używana między innymi do wyświetlania statystyk gracza oraz licznika FPS.
- Metoda zawiera takie argumenty jak:
  - x - położenie napisu na osi x
  - y - położenie napisu na osi y
  - text - tekst do wyświetlenia

Metoda addButton()

- Dodaje przycisk do listy przycisków obiektu o podanej nazwie przycisku, nazwie pliku, w podanym położeniu w oknie gry.
- Robiąc to tworzy nowy obiekt klasy Button wykorzystując do tego konstruktor, który rozmiar przycisku pobiera z pliku graficznego.

Metoda isCursorOnTheButton()

- Metoda sprawdza czy kursor myszy znajduje się na przycisku o podanej nazwie.
- Metoda przyjmuje argument buttonName - nazwa przycisku dla którego ma zostać wykonane sprawdzenie.
- Metoda zwraca prawdę jeśli kursor znajduje się na przycisku, fałsz w przeciwnym przypadku.

Metoda changeButtonTextureDependingCursorPosition()

- Metoda changeButtonTextureDependingCursorPosition wykorzystuje metodę isCursorOnTheButton do sprawdzania czy kursor myszy znajduje się nad przyciskiem, którego nazwę otrzymała przez parametr. W zależności od wyniku ustawia przyciskowi odpowiednią teksturę.
- Zapisz referencję przycisku o nazwie odebranej przez tą metodę jako parametr buttonName do zmiennej b.
- Jeśli kursor znajduje się nad sprawdzanym przyciskiem:
  - ustaw jego teksturę na tą, której nazwę metoda otrzymała jako parametr textureNameActive
- W przeciwnym przypadku:
  - Jeśli aktualna tekstura przycisku nie jest taka jaka powinna być w przypadku kiedy nie znajduje się nad nim kursor:
  - zmień teksturę na tą która reprezentuje przycisk ("niepodświetlony") nad którym nie znajduje się kursor.
- Nazwę tej tekstury metoda otrzymała jako parametr textureNameNormal.
- Metoda zawiera takie argumenty jak:
  - buttonName - nazwa sprawdzanego i/lub aktualizowanego przycisku.
  - textureNameActive - nazwa "podświetlonej" tekstury przycisku.

- textureNameNormal - nazwa "niepodświetlonej" tekstury przycisku.

Metoda getButton()

- Metoda odszukuje przycisk o podanej nazwie w liście przycisków i zwraca ten obiekt.

Metoda getMenu()

- Zwraca menu z listy o przekazanej do metody przez parametr nazwie.
- Metoda przyjmuje argument name.
- Metoda zwraca null.

Metoda draw()

- Rysuje przyciski z listy buttonList w oknie gry. (przyciski menu i wież)
- Pierwsza pętla rysuje główne menu.
- Druga pętla rysuje menu wyboru wież.

Metoda addButton()

- Metoda wykonuje to samo co metoda setButton() tyle tylko że jest publiczna więc wystarczy tamtą zmienić na publiczną albo tą usunąć.

Metoda quickAdd()

- Metoda upraszczająca dodawanie nowego przycisku do siatki przycisków menu.
- Metoda przyjmuje argument name oraz buttonTextureName.

Metoda setButton()

- Metoda obsługująca dodawanie przycisków w pozycji uzależnionej od tego, który z kolei jest to przycisk.
- W efekcie uzyskujemy siatkę przycisków u ustalonej liczbie przycisków w jednej linii.
- Każdy przycisk jest przesunięty względem innego o wielokrotność 64px.
- Przesuwa kolejna linię przycisków o 64px pomnożone przez numer linii
- Co optionsWidth przycisk z kolei, ustawia pozycje następnych przycisków na osi y niżej o iloczyn (64px + 16) i liczby przycisków podzielonych przez optionsWidth względem pierwszego przycisku przesuwa kolejny przycisk w prawo względem pierwszego o 64px na osi x pomnożone przez numer przycisku z kolei w tej linii

- Przyciskowi który został utworzony jako n-ty (buttonAmount) z kolei ustawia pozycję na osi x równą: pozycji początkowej menu + (numer przycisku z kolei % liczba przycisków w jednej linii) * (obliczony w konstruktorze odstęp dla tego menu + rozmiar kafelka (64px) + obliczony wcześniej odstęp odstęp dodany na końcu jest po to aby istniał odstęp również między pierwszym przyciskiem a początkiem menu.

Metoda isCursorOnTheButton()

- Metoda isCursorOnTheButton sprawdza czy kursor myszy znajduje się na przycisku o podanej nazwie.
- Metoda przyjmuje argument buttonName - nazwa przycisku dla którego ma zostać wykonane sprawdzenie.
- Metoda zwraca prawdę jeśli kursor znajduje się na przycisku, fałsz w przeciwnym przypadku.

Metoda isButtonClicked()

- Metoda sprawdza czy przycisk o podanej nazwie jest naciśnięty.
- Metoda przyjmuje argument buttonName
- Metoda zwraca prawdę jeśli przycisk jest naciśnięty, fałsz w przeciwnym przypadku.

Metoda getButton()

- Metoda odszukuje przycisk o podanej nazwie w liście przycisków i zwraca ten obiekt.

Metoda draw()

- Metoda rysująca przyciski.

6. Podział prac.

Maja Hendzel: AntimatterMissileLauncher.java, ProjectileAntimatterMissile.java, Disruptor.java, ProjectileDisruptor.java, LightWarhead.java, ProjectileLightWarhead.java, RocketLauncher.java, ProjectileRocket.java, HumanInterceptor.java, HumanHeavyFighter.java, HumanBomber.java, HumanAssaultShuttle.java, YuralInterceptor.java, YuralHeavyFighter.java, GremakInterceptor.java, GremakHeavyFighter.java, GremakBomber.java, GremakAssaultShuttle.java, TileType.java, TileGrid.java, Tile.java, ProjectileType.java, Player.java, MainMenu.java, Time.java.

Patrycja Kalita: sprawozdanie, Button.java, HeavyCannon.java, ProjectileHeavyCannon.java, LightAntimatterMissileLauncher.java, ProjectileLightAntimatterMissile.java, LightCannon.java, ProjectileLightCannon.java, Pulson.java, ProjectilePulson.java, PhidiInterceptor.java, PhidiHeavyFighter.java, PhidiBomber.java, PhidiAssaultShuttle.java, YuralBomber.java, YuralAssaultShuttle.java, OrthinInterceptor.java, OrthinHeavyFighter.java, OrthinBomber.java, OrthinAssaultShuttle.java, Wave.java, Unit.java, TowerType.java, Tower.java, Checkpoint.java, Map.java, Javadoc.

Marcin Kot: wygląd gry (grafiki przeciwników, wież, pocisków, broni, prycisków, tekstur, map), UI.java, HeavyWarhead.java, ProjectileHeavyWarhead.java, HellBore.java, ProjectileHellBore.java, MachineGun.java, ProjectileMachineGun.java, TwinCannon.java, TwinDisruptors.java, TwinHeavyWarhead.java, TwinLightCannon.java, ProjectileTwinCannon.java, TwinMachineGun.java, AshdakSpecialAce.java, AshdakInterceptor.java, AshdakHeavyFighter.java, AshdakBomber.java, AshdakAssaultShuttle.java, WaveManager.java, Projectile.java, Start.java, Game.java, Enemy.java, Editor.java, StateManager.java, Graphic.java, testy jednostkowe.

7. Wnioski.

W ramach naszego projektu wykonaliśmy grę „Tower Defense”. Wszystkie zaplanowane pracę, zostały wykonane.

Podczas wykonywania gry poszerzyliśmy swoją wiedzę w programowaniu obiektowym przy użyciu języka Java. Zapoznaliśmy się ze środowiskiem IntelliJ IDEA 2021.1 oraz poznaliśmy dwie nowe biblioteki: Slick2D i LWJGL.

W następnych etapach, gdybyśmy mieli rozwijąć grę to stworzylibyśmy m.in. ranking najlepszych graczy i stworzyli tą grę jako aplikację na telefon, dodalibyśmy więcej rodzajów wież, broni, pocisków oraz przeciwników.