Sprawozdanie Kamień milowy 2 :

1. Większość prezentacji znajduje się w formie wideo :

<https://youtu.be/8h0VGN73Zt0>

https://www.youtube.com/watch?v=8h0VGN73Zt0&feature=youtu.be

1. Zakres prac zrealizowanych :

* Adrian Kunecki:

Stworzenie grafik, mapy, tekstur oraz ich spakowanie.

Implementacja Kolizji

Stworzenie Klasy Ship, Main oraz Szkieletu B2WordlCreator

* Piotr Lepa:

Stworzenie klasy EnemyShip, EnemyBlaster, Blaster

Stworzenie modelu poruszania się Obiektów

Implementacja systemu odświeżenia list obiektów

* Prace wspólne:

Rozwój i implementacja metod i funkcji w klasie PlayScreen

Stworzenie MenuScreen i rozwój

1. Link do sklonowania projektu z gitHub:

https://github.com/ProjektyPSK/ProjecktGKLibGDX.git

1. Tematy nieomówione w prezentacji

* Czemu lista enemyBlaster jest statyczna?

W klasie EnemyShip znajduje się lista blasterów przeciwników

private static List<EnemyBlaster> *blasterList*

Jak wspominałem jest ona niezbędna do usuwania pocisków. W momencie Gdy dojdzie do strzału do właśnie tej listy jest dodawany blaster

*blasterList*.add(new EnemyBlaster(world, screen , this.getX() + this.getWidth() / 2, this.getY())) ;

W klasie PlayScreen jest użyta funkcja monitorująca listę blasterów, dzięki używania zmiennej statycznej, możemy dostać się do niej w sposób przedstawiony poniżej.

public void updateEnemyBlasterList () {  
 if (enemyShips.size > 0) {  
 for (EnemyBlaster blaster : EnemyShip.*getBlasterList*()) {  
 if (blaster.getStateTime() > 10 || blaster.isDestroyed() || blaster.isSetToDestroy()) {  
 enemyBlasterListToRemove.add(blaster);  
 }  
 }  
 for (EnemyBlaster blaster : enemyBlasterListToRemove) {  
 if (EnemyShip.*getBlasterList*().size() > 0)  
 EnemyShip.*getBlasterList*().remove(blaster);  
 }  
 }  
}

W innym przypadku konieczne byłoby iterowanie po liście przeciwników i po ich wewnętrznych listach blasterów, ale gdyby w ten sposób to wyglądało tego typu iterowanie musiałoby znajdować się w metodzie update i render, oraz w momencie zniszczenia statku nie byłoby możliwości wykonania tych metod na pociskach należących do statków, które już niknęły.

Jednocześnie warto wspomnieć o liście enemyBlasterListToRemove, na obecnie iterowanej liście nie można usuwać zmiennych stąd wynika konieczność użycia listy pomocniczej.

* możliwość przekroczenia wskazanego punktu

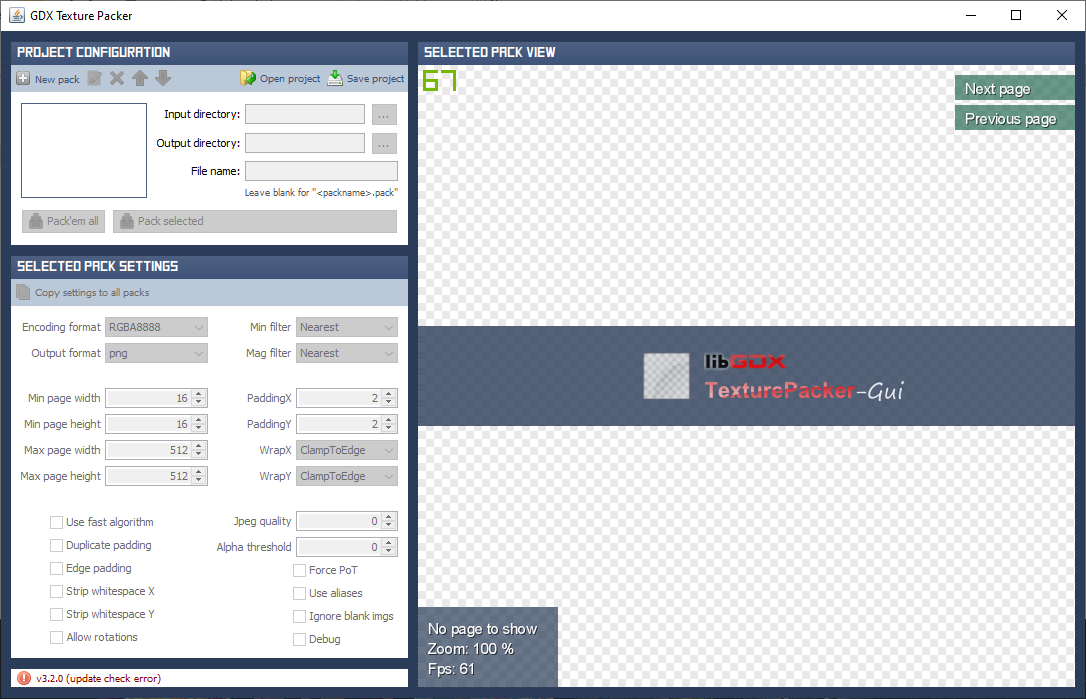
W klasie PlayScreen znajduje się metoda :

public void moveShipChecker(){  
 if (Math.*abs*( lastTouch - player.b2body.getPosition().x ) < 0.3){  
 player.b2body.setLinearVelocity(0,0);  
 }  
}

Więc moment sprawdzenia czy statek zbliżył się do punktu dotknięcia jest dosyć krótki w momencie szybkiego ruchu statku (im dłużej przytrzymamy statek tym szybciej statek się porusza) na podstawie powyższej metody to miejsce jest określane jako 30px (0.3\*100 ze względu na Main.PPM) na lewo i 30 px na prawo, jeśli wskazany dystans będzie pokonany szybciej niż 1/60 sekundy warunek nie zostanie wywołany, a pole sprawdzenia, nie może być większe bo byłoby zbyt zauważalne, że statek zatrzymuje się zbyt wcześnie.

* Pakowanie grafik

Plik z grafikami użyty w projekcie został stworzony za pośrenictwem edytora przeznaczonego do użytej biblioteki



1. Prace do realizacji w przyszłości:

* Poprawienie wizualne HUD i Menu (możliwe że wersja wizualna będzie się różnić wersji na gitHub i w prezentacji)
* Poprawienie gameplayu
* dodanie innych przeciwników (opcjonalne)
* Dodanie dźwięków do gry (opcjonalne)
* Dodanie nowych poziomów
* Zmiana systemu poruszania i kamery (opcjonalne)
* Tworzenie zapisów (opcjonalne)
* Zmiana lub dodanie dla niektórych przeciwników bardziej zaawansowanego poruszania się
* Dodanie animacji zmiany modelu (opcjonalne)
* Dodanie modyfikatorów rozgrywki