Uniwersytet Śląski

Interaktywne Aplikacje Multimedialne

Sprawozdanie do projektu:

Gra przeglądarkowa w języku JS

Wykonali:
Michał Jagieła
Rafał Nowak

Grupa:
PAW I

Spis treści

1. Wstęp

Treść zadania

Celem projektu było utworzenie gry przeglądarkowej przy użyciu elementu **Canvas HTML** oraz języka **JavaScript** lub stworzenie projektu przy użyciu biblioteki **Three.js** wykorzystywanej przy grafice 3D. Ostatecznie jako nasz projekt zdecydowaliśmy się utworzyć prostą grę przeglądarkową. Gra powstała na zasadzie **„endless dungeon crawler”**, oznacza to że jej rozgrywka polega cyklu:

wejdź do pokoju -> walcz z przeciwnikiem -> przejdź do następnego pokoju

Gra polega więc na przejściu jak największej ilości pokoi. W momencie śmierci naszej postaci następuje koniec gry.

Motywacja

Głównym motywem wyboru takiej tematyki gry jest jej prostota. Gra tego gatunku nie wymaga rozbudowanej fabuły a poziomy mogą się powtarzać. Dzięki temu nie musieliśmy zaplanować całego świata gry a jedynie jeden poziom z różnymi przeciwnikami. Dodatkową zaletą jest to że nie musieliśmy statycznie przypisać określonego przeciwnika do określonego pokoju. Zamiast tego każdy przeciwnik jest losowany. Jedynym wyjątkiem jest pojawienie się co 10 poziomów znacznie silniejszego przeciwnika.

2. Specyfikacja zewnętrzna

Na początku po uruchomieniu gry pojawia się ekran startowy. W nim mamy wybór rozpoczęcia nowej gry lub kontynuowanie poprzednio zapisanej. Poruszanie się po podanych opcjach następuje na dwa sposoby:

1) Poprzez wykorzystanie myszki, gdzie **LPM** służy za wybór opcji
2) Poprzez wykorzystanie klawiszy klawiatury **Strzałka w górę, Strzałka w dół** do poruszania oraz klawisza **Enter** do wyboru opcji

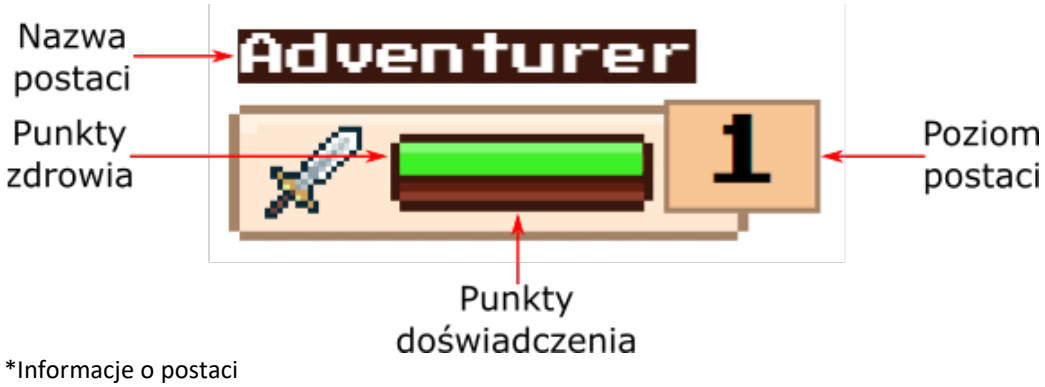
Wybór opcji kontynuacji gry jest widoczny tylko wtedy gdy użytkownik posiada zapis rozgrywki.

*Ekran startowy gry, bez zapisu

*Ekran startowy gry z dostępnym zapisem

Po wybraniu jednej z dostępnych opcji ukazuję się właściwy ekran gry. Widoczna staje się nasza postać, pokój w której się ona znajduje oraz element interfejsu przedstawiający takie rzeczy jak **nazwa postaci**, **pasek zdrowia**, **pasek punktów doświadczenia**, **poziom postaci**.

*Ekran gry z widoczną postacią i elementem interfejsu



*Informacje o postaci

Poruszanie się po ekranie gry jest możliwe poprzez wykorzystanie klawiszy **Strzałka w lewo**, **Strzałka w prawo**. Naciśnięcie klawisza **Esc** spowoduje pojawienie się menu pauzy zawierające opcje zapisu oraz zamknięcia gry. Podobnie jak na ekranie tytułowym, poruszanie się po menu pauzy jest możliwe na dwa sposoby:

1) Poprzez wykorzystanie myszki, gdzie **LPM** służy za wybór opcji
2) Poprzez wykorzystanie klawiszy klawiatury **Strzałka w górę, Strzałka w dół** do poruszania oraz klawisza **Enter** do wyboru opcji

Aby zamknąć menu pauzy należy ponownie nacisnąć klawisz **Esc**.

*Ekran gry z widocznym menu pauzy

Przy próbie wyjścia z pokoju w lewą stronę zostaniemy zatrzymani. Na dole ekranu pojawi się wiadomość tekstowa. W czasie jej tworzenia możemy nacisnąć przycisk **Next** poprzez użycie **LPM** lub klawisza **Enter**. Dzięki temu wiadomość natychmiastowo skończy się pisać. Po wypisaniu pełnej wiadomości możemy ponownie nacisnąć przycisk **Next** do zamknięcia jej.

*Ekran gry z widoczną wiadomością

W momencie przejścia na środek pokoju rozpocznie się sekwencja walki. Na początek pojawi się przeciwnik a następnie zostanie wyświetlona wiadomość. Następnie po kliknięciu przycisku **Next** przeciwnik zajmie pozycje a następnie zacznie się bitwa począwszy od tury gracza. Zmiany nastąpią w interfejsie. Między innymi informacje o graczu zostaną powiększone oraz pojawią się informacje o przeciwniku. W trakcie tur gracza widoczne jest także menu wyboru akcji.

*Ekran gry z widoczną wiadomością

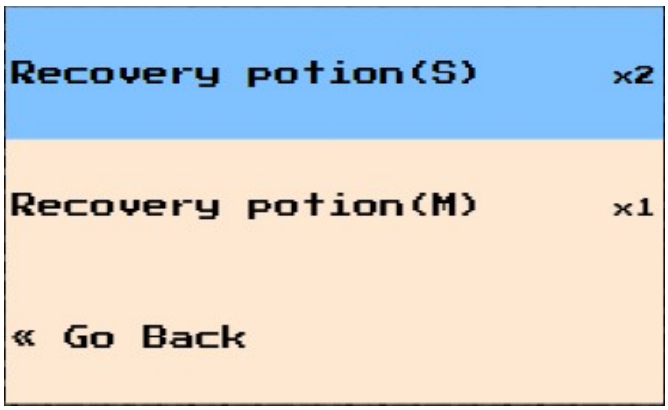
*Ekran bitwy podczas tury gracza

Menu akcji posiada możliwość ataku oraz użycia przedmiotu. Poruszanie się po menu akcji następuje na dwa sposoby:

1) Poprzez wykorzystanie myszki, gdzie **LPM** służy za wybór opcji
2) Poprzez wykorzystanie klawiszy klawiatury **Strzałka w górę, Strzałka w dół** do poruszania oraz klawisza **Enter** do wyboru opcji

Zarówno opcja ataku jak i przedmiotów posiada więcej opcji zagnieżdżonych w sobie. Każda akcja posiada opis wyświetlany na dole ekranu.

*Wybór ataku z menu akcji



*Wybór przedmiotu z menu akcji

W obecnej implementacji gry, gracz na start posiada trzy rodzaje ataków:

- **Fast attack** – podstawowy atak zadający średnie obrażenia
- **Heavy attack** – atak zadający duże obrażenia przeciwnikowi oraz mniejsze postaci używającej go
- **Hidden blade** – atak zadający małe obrażenia ale zatruwa przeciwnika

Oprócz tego gracz rozpoczyna grę z trzema przedmiotami przywracającymi zdrowie.

Po wybraniu jednej z akcji, zostaje ona wykonana a następnie kończy się tura gracza i zaczyna przeciwnika. Następnie przeciwnik decyduje o wyborze akcji i cykl się powtarza.

Obecnie gra posiada zaimplementowane dwa statusy nakładane na postać:

- **Poison** – zatrucie. Zadaje obrażenia zależnie od ilości maksymalnych punktów życia zatrutej postaci. Działa przez 2 tury.
- **Fire** – podpalenie. Zadaje obrażenia zależnie od punktów ataku postaci nakładającej status. Działa przez 2 tury.

Walka polega na turach postaci następujących po sobie i skończy się dopiero wtedy gdy punkty życia jednej z postaci spadną do wartości mniejszej lub równej 0. W przypadku wygrania walki przez gracza, otrzymuje on punkty doświadczenia potrzebne do zwiększenia poziomu a co za tym idzie statystyk postaci. Ilość otrzymanych punktów doświadczenia zależy od typu przeciwnika oraz jego poziomu.

*Ekran bitwy z widoczną ilością otrzymanych punktów doświadczenia

W przypadku wygranej walki, utworzony zostanie zapis gry. W przypadku przegranej bitwy istniejący zapis zostanie usunięty a następnie pojawi się ekran śmierci.

*Ekran śmierci

3. Specyfikacja wewnętrzna

Klasy związane z systemem bitwy

Battle
Klasa odpowiedzialna za system bitw. Odpowiada m.in. za dodawanie postaci do bitwy, śledzenie przedmiotów użytych podczas bitwy.

- **addCombatant()** – Funkcja dodająca walczące postacie do bitwy. Przyjmowanymi argumentami są: **identyfikator postaci**, **drużyna postaci**, **statystyki postaci**. Obiekty typu **Combatant** są wpisywane do tablicy **combatants**.
- **createElement()** – Funkcja przygotowująca ekran bitwy
- **init()** – Funkcja inicjalizująca bitwę. Przyjmowanym parametrem jest **kontener HTML** w którym ma się znaleźć ekran bitwy. Funkcja na początku wstawia element ekranu bitwy w element nadrzędny będący polem ekranu gry. Następnie każda z postaci z tablicy **combatants** zostaje zainicjalizowana. Na koniec funkcja tworzy nowy obiekt klasy **TurnCycle** a następnie go inicjalizuje.

BattleEvent
Klasa odpowiedzialna za zdarzenia w trakcie bitwy.

- **textMessage()** – Funkcja odpowiedzialna za tworzenie wiadomości. Jako parametr przyjmuje funkcje **resolve()**.
- **stateChange()** – Funkcja odpowiedzialna za zmianę stanu postaci. Jako parametr przyjmuje funkcje **resolve()**. Funkcja odpowiedzialna jest za zadawanie obrażeń postaciom, leczenie oraz nakładanie i usuwanie statusów.
- **submissionMenu()** – Funkcja odpowiedzialna za wywołanie Manu wyboru akcji. Jako parametr przyjmuje funkcje **resolve()**. Wewnątrz funkcji tworzony jest nowy obiekt klasy **SubmissionMenu** a następnie jest on inicjalizowany.
- **giveXp()** – Funkcja odpowiedzialna za rozdanie punktów doświadczenia po pokonaniu przeciwnika. Jako parametr przyjmuje funkcje **resolve()**. Wewnątrz funkcji uruchamiana jest animacja wypełnienia paska doświadczenia. W przypadku przekroczenia maksymalnej ilości doświadczenia zostaje zwiększony poziom postaci.
- **animation()** – Funkcja wywołująca wybraną animacje. Jako parametr przyjmuje funkcje **resolve()**.
- **init()** – Funkcja inicjalizująca zdarzenie. Jako parametr przyjmuje funkcje **resolve()**. Wewnątrz następuje wywołanie jednej z funkcji zawartych w klasie.

Combatant
Klasa odpowiedzialna za obsługę informacji o walczących postaci.

- **get hpPercent()** – Getter zwracający procentową wartość punktów zdrowia.
- **get xpPercent()** – Getter zwracający procentową wartość punktów doświadczenia.
- **get givesXp()** – Getter zwracający ilość punktów doświadczenia zdobywanych za zabicie przeciwnika
- **createElement()** – Funkcja przygotowująca elementy interfejsu bitwy
- **update()** – Funkcja aktualizująca interfejs bitwy. Przyjmowanym parametrem jest obiekt zawierający parametry ulegające zmianie.
- **getStatusEvents()** – Funkcja zwracająca zdarzenia związane z aktywnymi statusami
- **statusTurnDecrement()** – Funkcja zmniejszająca czas pozostałych tur działania statusu
- **init()** – Funkcja inicjalizująca. Przyjmowanym parametrem jest **kontener HTML** w którym mają znaleźć się informacje.

SubmissionMenu
Klasa odpowiedzialna za obsługę menu wyboru akcji.

- **getPages()** – Funkcja zwracająca zawartość menu wyboru akcji.
- **menuSubmit()** – Funkcja odpowiadająca za wybór akcji. Przyjmowanymi argumentami są: **wybrana akcja**, **identyfikator przedmiotu**(w przypadku gdy został użyty)
- **decide()** – Funkcja losująca akcję przeciwnika
- **showMenu()** – Funkcja wyświetlająca menu wyboru akcji. Przyjmowanym parametrem jest **kontener HTML** w którym mają ono znaleźć.
- **init()** – Funkcja inicjalizująca. Przyjmowanym parametrem jest **kontener HTML.**

TurnCycle
Klasa odpowiedzialna za obsługę cyklu tur walki.

- **turn()** – Asynchroniczna funkcja tury. Na początku sprawdzane jest czyja jest tura, następnie wywołane zostaje menu wyboru akcji. W przypadku użytego przedmiotu zostaje on usunięty z listy dostępnych przedmiotów. Następnie zostają wywołane zdarzenia związane z powodzeniem akcji. Po nich następuje wywołanie efektów statusu o ile jest on aktywny. Na koniec sprawdzane jest czy któraś ze stron wygrała walkę. Jeżeli nie to następuje zmiana aktywnej strony i funkcja zostaje ponownie wywołana.
- **getXpFromEnemy()** – Asynchroniczna funkcja wywołująca zdarzenie pobrania punktów doświadczenia z pokonanego przeciwnika. Przyjmowanym parametrem jest zmienna **submission** reprezentująca wybór akcji z menu.
- **getWinningSide()** – Funkcja sprawdzająca czy któraś ze stron nie żyje

Klasy związane z obiektami gry

GameObject
Klasa odpowiedzialna za zarządzanie wszelkimi obiektami w grze

Person
Klasa dziedzicząca po **GameObject**. Odpowiada za zarządzanie rysowanymi postaciami

- **update()** – Funkcja odpowiedzialna za wprowadzanie zmian w obiekcie. Przyjmowanym parametrem jest **stan gry**.
- **updatePosition()**– Funkcja obsługująca zmianę koordynatów obiektu.
- **updateSprite()** – Funkcja obsługująca zmianę animacji obiektu.
- **startBehavior()** – Funkcja wywołująca określone zachowanie obiektu. Przyjmowanymi argumentami są: **stan gry, zachowanie**.

EnemySlime
Klasa dziedzicząca po **GameObject**. Odpowiada za zarządzanie rysowanymi przeciwnikami

- **update()** – Funkcja odpowiedzialna za wprowadzanie zmian w obiekcie. Przyjmowanym parametrem jest **stan gry**.
- **updatePosition()**– Funkcja obsługująca zmianę koordynatów obiektu.
- **updateSprite()** – Funkcja obsługująca zmianę animacji obiektu.
- **startBehavior()** – Funkcja wywołująca określone zachowanie obiektu. Przyjmowanymi argumentami są: **stan gry, zachowanie**.
- **scaleEnemy()** – Funkcja zmieniająca statystyki przeciwnika zależnie od poziomu gracza. Przyjmowanymi argumentami są: **podstawowe statystyki przeciwnika, poziom gracza**.

Sprite
Klasa odpowiadająca za rysowanie obiektów

- **get frame()** – Getter zwracający obecną klatkę do narysowania
- **setAnimation()** – Funkcja ustawiająca aktywną animację. Przyjmowanym parametrem jest **nazwa sekwencji animacji**.
- **setTurnAnimation()** – Funkcja ustawiająca aktywną animację w trakcie tury walki. Przyjmowanym parametrem jest **nazwa sekwencji animacji**. W przeciwieństwie do **setAnimation()** animacja wywołana tą funkcją wykona się tylko raz.
- **updateAnimationProgress()** – Funkcja aktualizująca postęp animacji.
- **draw()** – Funkcja rysująca obiekt. Przyjmowanym parametrem jest **kontekst canvas**.

Klasy związane z menu gry

KeyboardMenu
Klasa odpowiadająca za menu wyboru

- **setOptions()** – Funkcja wypełniająca menu opcjami. Przyjmowanym parametrem jest **tablica obiektów opcji**.
- **createElement()** – Funkcja przygotowująca menu.
- **end()** – Funkcja zamykająca menu wyboru.
- **init()** – Funkcja inicjalizująca. Przyjmowanym parametrem jest **kontener HTML.** Funkcja wpisuje zawartość menu do kontenera a następnie sprawdza czy po naciśnięciu **strzałki w dół** lub **strzałki w górę** występuje niewyłączona opcja. Jeżeli tak to na nią się przełącza.

PauseMenu
Klasa odpowiadająca za menu pauzy

- **getOptions()** – Funkcja zwracająca zawartość menu.
- **createElement()** – Funkcja przygotowująca menu.
- **close()** – Funkcja zamykająca menu pauzy.
- **init()** – Asynchroniczna funkcja inicjalizująca. Przyjmowanym parametrem jest **kontener HTML.**

GameOverMenu
Klasa odpowiadająca za ekran śmierci

- **getOptions()** – Funkcja zwracająca zawartość menu.
- **createElement()** – Funkcja przygotowująca menu.
- **close()** – Funkcja zamykająca ekran śmierci.
- **init()** – Asynchroniczna funkcja inicjalizująca. Przyjmowanym parametrem jest **kontener HTML.**

TitleScreen
Klasa odpowiadająca za ekran tytułowy

- **getOptions()** – Funkcja zwracająca zawartość menu. Przyjmowanym parametrem jest funkcja **resolve()**.
- **createElement()** – Funkcja przygotowująca menu.
- **close()** – Funkcja zamykająca ekran tytułowy.
- **init()** – Asynchroniczna funkcja inicjalizująca. Przyjmowanym parametrem jest **kontener HTML.**

Klasy kontrolujące grę

Overworld
Klasa przyjmująca rolę kontrolera w którym zamieszczone są pozostałe komponenty

- **startGameLoop()** – Funkcja odpowiedzialna za rozpoczęcie pętli będącą podstawą działania gry.
- **bindActionInput()** – Funkcja przypisująca akcje do przycisku.
- **bindHeroPositionCheck()** – Funkcja przypisująca nasłuchiwanie ruchu gracza w celu szukania zdarzenia znajdującego się w pomieszczeniu.
- **start map()** – Funkcja wczytująca poziom. Przyjmowanymi parametrami są: **konfiguracja poziomu** oraz **wczytany stan**(tylko po załadowaniu zapisu). Funkcja na początku generuje naszego bohatera oraz przeciwnika. Następnie na podstawie wygenerowanego przeciwnika tworzone jest zdarzenie rozpoczęcia bitwy w odpowiednim miejscu pomieszczenia. Jeżeli istnieje zapis to następuje nadpisanie wcześniej wygenerowanych wartości.
- **init()** – Asynchroniczna funkcja inicjalizująca. Wewnątrz niej jest ustawiany główny **kontener HTML** gry, uruchamiany jest ekran startowy, wczytywany jest zapis, inicjalizowany jest interfejs użytkownika, wczytywany jest poziom, przypisane są klawisze akcji oraz nasłuch zdarzeń, inicjalizowany jest nasłuch klawiszy ruchu i ostatecznie uruchamiana jest pętla gry.

OverworldEvent
Klasa odpowiadająca za zdarzenia globalne.

- **stand()**, **walk()**, **long walk()** – Funkcja wywołujące zachowanie postaci na mapie. Przyjmują jako parametr funkcje **resolve()**.
- **textMessage()** – Funkcja wywołująca wyświetlenie wiadomości. Przyjmuje jako parametr funkcję **resolve()**.
- **changeRoom()** – Funkcja wywołująca zmianę pomieszczenia. Przyjmuje jako parametr funkcję **resolve()**.
- **battle()** – Funkcja wywołująca rozpoczęcie walki. Przyjmuje jako parametr funkcję **resolve()**.
- **modifyPlayerFlag()** – Funkcja zmieniająca wartość globalnej flagi gracza. Przyjmuje jako parametr funkcję **resolve()**.
- **pause()** – Funkcja wyświetlenia menu pauzy. Przyjmuje jako parametr funkcję **resolve()**.
- **gameOver()** – Funkcja wyświetlenia menu śmierci. Przyjmuje jako parametr funkcję **resolve()**.

- **init()** – Funkcja inicjalizująca. Wywołuje jedną z funkcji zawartych w klasie **OverworldEvent**.

OverworldMap
Klasa odpowiadająca za mapę gry.

- **drawImage()** – Funkcja rysująca tło poziomu. Przyjmowanym parametrem jest **kontekst canvas**.
- **mountObjects()** – Funkcja odpowiedzialna za przypisanie identyfikatorów obiektom znajdującym się na mapie.
- **startCutscene()** – Asynchroniczna funkcja odpowiedzialna za uruchomienie przerywnika filmowego. Jako parametr przyjmuje **tablice obiektów zdarzeń**.
- **checkForCutscene()** – Funkcja sprawdzająca czy w danym miejscu ma zostać wywołane zdarzenie.

Klasy nasłuchujące klawiaturę

DirectionInput
Klasa odpowiadająca za nasłuchiwanie klawiszy ruchu postaci

- **get direction()** – Getter zwracający tablicę przyciśniętych przycisków
- **init()** – Funkcja inicjalizująca. Dodaje nasłuchiwanie przyciśnięcia i puszczenia przycisku.

KeyPressListener
Klasa odpowiadająca za nasłuchiwanie klawiszy

- **unbind()** – Funkcja wyłączająca nasłuch przycisku

Klasy obsługi zapisu

SaveManagement
Klasa odpowiadająca za zarządzanie zapisami

- **save()** – Funkcja zapisu gry. Wykorzystuje **localStorage** przeglądarki do przechowania zapisu.
- **deleteSave()** – Funkcja usunięcia zapisu.
- **getSaveFile()** – Funkcja znajdująca zapis w **localStorage**.
- **load()** – Funkcja wczytująca zapis.

Klasy wyświetlające tekst

TextMessage
Klasa odpowiadająca za system wyświetlenia wiadomości

- **createElement()** – Funkcja przygotowująca wiadomość.
- **done()** – Funkcja odpowiedzialna za przyśpieszenie wyświetlania tekstu lub zamknięcie go.
- **init()** – Funkcja inicjalizująca. Jako parametr przyjmuje **kontener HTML**.

RevealingText
Klasa odpowiadająca za efekt pisania litera po literze

- **revealOneCharacter()** – Funkcja wyświetlająca literę. Jako parametr przyjmuje **tablice znaków**.
- **warpToDone()** – Funkcja przyśpieszająca wyświetlenie wiadomości.
- **init()** – Funkcja inicjalizująca.

Pozostałe klasy

Hud
Klasa odpowiadająca za zarządzanie interfejsem gracza

- **update()** – Funkcja przygotowująca za aktualizację informacji o graczu.
- **createElement()** – Funkcja przygotowująca interfejs gracza.
- **init()** – Funkcja inicjalizująca. Jako parametr przyjmuje **kontener HTML**.

SceneTransition
Klasa odpowiadająca za efekt wygaszenie podczas przechodzenia między pokojami

- **createElement()** – Funkcja przygotowująca efekt przejścia.
- **fadeOut()** – Funkcja wywołująca animacje wygaśnięcia.
- **init()** – Funkcja inicjalizująca. Jako parametr przyjmuje **kontener HTML** oraz **funkcję zwrotną**.

4. Podsumowanie

Wnioski

W trakcie tworzenia projektu, doszliśmy do wniosku, iż język **JavaScript** w połączeniu z elementami **canvas HTML**, przy odpowiedniej ilości czasu, jest wystarczający do stworzenia prostej gry przeglądarkowej. Decydując się na użycie dodatkowych bibliotek tworzenie gry staję się łatwiejsze, a sama gra może zostać jeszcze bardziej rozwinięta, bez porównania z grą zbudowaną w oparciu o sam język **JavaScript** oraz **elementy HTML**.

Możliwości rozwoju

W obecnym stanie gra posiada ogromną ilość ścieżek rozwoju. Początkowo należałoby zająć się kwestią dodania odpowiednio pasującej ścieżki audio, następnie zwiększenie różnorodności przeciwników, większej ilości ataków, gdzie nauczenie się ich wymaga zdobycie kolejnego poziomu.

W dalszym rozwoju widzimy dodanie systemu tworzenia przedmiotów. Polegałby on zbieraniu materiałów z przeciwników i otoczenia, aby następnie je użyć w celu stworzenia przedmiotów użytkowych które można by użyć w trakcie walk.

Dodatkowo można zwiększyć personalizacje naszego bohatera. Obecnie początkowe statystyki są wpisane na sztywno a następnie skalują się wraz z poziomem. Personalizacja bohatera oznacza to, że przy rozpoczęciu gry mielibyśmy dostęp do rozdysponowania punktów statystyk według naszego uznania. Następnie przy zdobyciu nowego poziomu otrzymalibyśmy określoną ilość punktów które moglibyśmy wykorzystać do zwiększenia statystyk.

Oprócz przedmiotów użytkowych można by wprowadzić przedmioty noszone przez gracza. Wprowadziło by to do gry system ekwipunku, a noszone przedmioty miałyby dalszy wpływ na statystyki gracza. Co za tym idzie można by wprowadzić system sklepów w których można kupować i sprzedawać przedmioty posiadane przez nas.