Kierunek: ***Informatyka***

2025/2026

Dominik Gumola

PRACA INŻYNIERSKA

Projekt i implementacja gry strategicznej z systemem zarządzania zasobami i ekwipunkiem

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, *2026*

Spis treści

1. Wstęp

Współczesny rynek gier komputerowych charakteryzuje się dynamicznym rozwojem oraz coraz większą różnorodnością gatunków i platform docelowych. Gry strategiczne od wielu lat stanowią jeden z kluczowych segmentów branży, oferując graczom bardziej analityczne podejście do rozgrywki. Ich głównym założeniem jest stawianie gracza w roli decydenta, którego zadaniem jest planowanie działań, zarządzanie zasobami oraz podejmowanie decyzji mających bezpośredni wpływ na przebieg rozgrywki i osiągnięcie określonych celów. W przeciwieństwie do gier zręcznościowych, sukces w grach strategicznych w większym stopniu zależy od umiejętności analitycznych, logicznego myślenia oraz przewidywania konsekwencji podejmowanych decyzji. Istotnym elementem gier strategicznych jest zarządzanie zasobami, które mogą obejmować jednostki, surowce, czas, energię lub przestrzeń. Gracz zobowiązany jest do optymalnego wykorzystania dostępnych zasobów w warunkach ograniczeń narzuconych przez mechaniki gry. Często decyzje podejmowane na wczesnym etapie rozgrywki mają długofalowy wpływ na jej dalszy przebieg, co zwiększa znaczenie planowania oraz strategii długoterminowej.

1.1. Gry strategiczne typu czasu rzeczywistego

Strategie czasu rzeczywistego charakteryzują się dynamiczną rozgrywką, w której wszystkie działania gracza wykonywane są w czasie rzeczywistym. Jednym z kluczowych czynników wpływających na skuteczność rozgrywki w tego typu grach jest wskaźnik APM określający liczbę akcji wykonywanych przez gracza w ciągu jednej minuty. Wysoka wartość APM jest często niezbędna do efektywnego zarządzania jednostkami, ekonomią oraz reagowania na działania przeciwnika. W praktyce oznacza to konieczność jednoczesnego wykonywania wielu czynności, takich jak wydawanie rozkazów bojowych, zarządzanie produkcją jednostek, kontrola mapy oraz mikrozarządzanie poszczególnymi oddziałami. Dla doświadczonych graczy stanowi to istotny element rywalizacji i umiejętności, jednak dla osób początkujących lub nie poświęcających znacznej części

swojego czasu na praktykowanie mechanik takich gier może być barierą wejścia. Wysokie wymagania dotyczące refleksu oraz koordynacji ruchowej sprawiają, że gry RTS są postrzegane jako gatunek wymagający dużego nakładu czasu na naukę i trening.

1.2. Gry strategiczne typu turowego

W przeciwieństwie do podgatunku strategii czasu rzeczywistego, gry podgatunku strategii turowej eliminują presję czasu, umożliwiając graczowi podejmowanie decyzji w sposób bardziej przemyślany i analityczny. Rozgrywka oparta na turach pozwala na zatrzymanie akcji, analizę sytuacji na planszy oraz zaplanowanie kolejnych działań bez konieczności szybkiego reagowania. Taki model rozgrywki jest znacznie bardziej przyjazny dla graczy, których czas jaki mogą poświęcić na grę jest ograniczony lub preferują spokojniejsze tempo rozgrywki. Brak wymagań związanych z wysokim APM sprawia, że gry turowe koncentrują się przede wszystkim na podejmowaniu decyzji strategicznych, a nie na zręczności i praktyki gracza. Dzięki temu są one bardziej dostępne dla szerszego grona odbiorców, w tym osób rozpoczynających swoją przygodę z grami strategicznymi.

1.3. Kombinacja strategii turowej oraz czasu rzeczywistego

Gry typu *auto-battle* stanowią współczesną ewolucję gatunku strategicznego, łączącą elementy planowania znane z gier turowych z automatyzacją kluczowych etapów rozgrywki. W tego typu grach rola gracza skupia się głównie na fazie przygotowania, obejmującej dobór jednostek, zarządzanie ekwipunkiem oraz optymalizację ustawienia przed rozpoczęciem walki. Sam przebieg starcia realizowany jest automatycznie przez system gry, bez potrzeby bieżącej ingerencji ze strony gracza.

Takie podejście znacząco redukuje wymagania dotyczące APM, redukując współczynnik który mógłby odrzucić potencjalnych graczy pozwalając jednak na doświadczenie bitew które towarzyszą temu gatunkowi gier. Gracz może skupić się na aspektach strategicznych i decyzyjnych, bez presji czasu i konieczności wykonywania dużej liczby akcji w krótkim przedziale czasowym.

Obniżenie progu wejścia poprzez ograniczenie wymagań związanych z APM ma szczególne znaczenie w kontekście gier mobilnych. Urządzenia z ekranem dotykowym nie sprzyjają precyzyjnemu sterowaniu charakterystycznemu dla gier strategii czasu rzeczywistego, co dodatkowo wzmacnia popularność gier turowych oraz *auto-battlerów* na platformie Android. Dzięki automatyzacji procesów takich jak rozgrywana bitwa oraz uproszczony interfejs, gry te umożliwiają prowadzenie rozgrywki w krótszych sesjach, dostosowanych do codziennego użytkowania urządzeń mobilnych. W efekcie gry strategiczne oparte na modelu turowym oraz *auto battle* stanowią przystępniejszą alternatywę dla klasycznych stratgeii, oferując równowagę pomiędzy głębią strategiczną a dostępnością dla szerokiego grona odbiorców.

1.4. Cel pracy

Celem niniejszej pracy inżynierskiej jest zaprojektowanie oraz implementacja gry strategicznej z elementami zarządzania ekwipunkiem. Projekt zakłada stworzenie funkcjonalnego prototypu gry, który umożliwia graczowi podejmowanie decyzji strategicznych poprzez dobór przedmiotów, jednostek oraz ich rozmieszczenie przed rozpoczęciem starcia z przeciwnikiem. Projekt stanowił również okazję do praktycznego zastosowania wiedzy zdobytej podczas studiów, w szczególności w zakresie programowania obiektowego, projektowania systemów gry oraz tworzenia aplikacji mobilnych.

1.5. Motywacja

Czynnikami motywującymi była chęć zaprojektowania gry dostosowanej do specyfiki urządzeń mobilnych, uwzględniającej ograniczenia sprzętowe oraz potrzeby użytkowników systemu Android, a także implementacja gry pozbawionej popularnych w ostatnich latach mechanik ograniczających czas rozgrywki, z naciskiem na możliwość gry w trybie offline.

2. Analiza biznesowa

Projekt gry skupia się na podgatunku o anglojęzycznej nazwie *auto-battle* oraz dostarczeniu prostej rozgrywki dla pojedynczego gracza, której celem będzie zarządzanie dostępnymi zasobami tak by przetrwać jak najdłużej. Kluczowym elementem rozgrywki jest zarządzanie oraz przetwarzanie zasobów dostępnych użytkownikowi. by móc dłużej opierać się natarciom ze strony wroga.

2.1. Analiza rynku i konkurencji

Na rynku znajduje się wiele gier strategicznych. Przyjęty dla projektu gatunek *auto-battle* skupia się głównie na mechanice wystawienia przez gracza drużyny oraz zwiększaniu jej siły poprzez łączenie duplikatów poszczególnych rodzajów jednostek prowadząc tym samym do zwiększenia jej poziomu. Wiele gier skupia się głównie na tym elemencie automatycznej walki drużyn gracza i wroga, przykładem jest *Art of war: Legions* lub *Teamfight: Tactics*.

Inne pomijają element łączenia duplikatów na rzecz dodania obozu pozwalającego na modyfikowanie sił gracza poza ekranem bitwy przykładem jest *Dawn of Ages*.
Rynek oferuje również tytuły łączące oba powyższe podejścia w jedno. W takich grach gracz może ulepszać siłę swojej armii na dwa sposoby, przy pomocy obozu jaki i duplikatów podczas bitwy. Przykładem jest *King God Castle*.

Niestety, gry o takiej formule mają poważną wadę. Jeśli nie są nastawione na aktywną rozgrywkę wieloosobową, pojawia się istotny problem. Tempo progresu w tych grach jest często sztucznie ograniczane przy pomocy *walut* lub *zasobów*. Odnawiają się one dopiero po upływie kilku godzin lub nawet dni. Zużywane są natomiast podczas samej rozgrywki, przez co graczowi często blokowana jest możliwość dalszego postępu. W efekcie gracz musi czekać na odnowienie tych zasobów, aby móc kontynuować grę. W grach tego typu jedyną metodą na obejście ograniczeń jest zakup zasobów w dedykowanym do tego celu sklepie z mikropłatnościami, a system ten niestety nie pozwala grać bez połączenia z internetem.
Projektowana gra ma na celu połączenie mechanik obozu i auto bitwy pozwalając użytkownikowi na cieszenie się konceptem rozgrywki niewymagającego nadmiernego poświęcania czasu na naukę mechanik.

2.2. Wymagania funkcjonalne

Gra musi posiadać przejrzysty i intuicyjny interfejs użytkownika, umożliwiający komfortowe zarządzanie zasobami, obozem oraz drużyną gracza. Rozgrywka przebiega w ramach czterech głównych scen, między którymi przejścia realizowane są poprzez interakcję z interfejsem oraz sekwencyjne podejmowanie decyzji.

- **Interfejs użytkownika -** Gra musi posiadać dostęp do przejrzystego i intuicyjne zrozumiałego interfejsu pozwalającego na zarządzanie zasobami, obozem i drużyną gracza tak by podejmowanie decyzji było komfortowe.
- **Rozgrywka/mechanika gry -** Gra rozgrywa się między 4 scenami między którymi przejścia zapewnione są dzięki interfejsowi użytkownika.
    - Scena startowa obozu pozwalająca na wydawanie rozkazów odnośnie rozbudowy i przetwarzania przedmiotów.
    - Scena podsumowująca mapy świata, będąca następstwem sceny obozu po zakończeniu w nim działań. Pozwala ona na spojrzenie na teren wokół obozu oraz umożliwia zaatakowanie grup wroga znajdujących się na okolicznych terenach. Scena ta pozwala na na zakończenie cyklu wracając do sceny skupiającej się na obozie lub w przypadku podjęcia walki z wrogiem przejście do sceny przed walką.
    - Scena przygotowania do bitwy po zdecydowaniu o podjęciu walki w scenie poprzedniej gracz ma możliwość ustawienia liczebności, pozycji oraz ekwipunku drużyny przed faktycznym rozpoczęciem walki.
    - Scena bitwy będąca następstwem sceny przygotowawczej, gdzie odbywa się symulacja walki drużyn gracza i jego przeciwnika. Po zakończeniu symulacji gracz powraca do sceny obozu jeśli udało mu się pokonać grupę przeciwnika.

2.3. Wymagania niefunkcjonalne

Aplikacje gier komputerowych tworzone są z wydajnością jako głównym elementem. Bez nacisku na ten element produkt tej kategorii z perspektywy odbiorcy jest niezdatny do użytku. Każda aplikacja musi spełniać to wymaganie, przez naturę działania gier komputerowych wydajność i optymalizacja kodu źródłowego jest krytycznym i głównym wymaganiem potrzebnym dla zapewnienia wysokiej jakości aplikacji.

- **Kompatybilność z systemami Android –** Gra powinna działać poprawnie na urządzeniach mobilnych z systemem Android, uwzględniając różne wersje systemu, rozdzielczości ekranów oraz konfiguracje sprzętowe. Aplikacja musi zapewniać prawidłową obsługę orientacji ekranu i dostępnych zasobów systemowych.
- **Tryb offline –** Gra musi być funkcjonalna mimo braku połączenia internetowego. Wszystkie podstawowe mechaniki, takie jak sterowanie, logika gry, animacje i zapisywanie stanu gry, powinny działać bez dostępu do sieci, a dane niezbędne do rozgrywki powinny być przechowywane lokalnie na urządzeniu.
- **Wydajność i optymalizacja –** Gra musi być napisana w sposób umożliwiający płynną i wydajną rozgrywkę na szerokim zakresie urządzeń mobilnych. Obejmuje to optymalizację zużycia pamięci RAM i procesora, minimalizowanie opóźnień w renderowaniu grafiki 2D, a także efektywne zarządzanie zasobami, takimi jak tekstury, dźwięki i animacje.
- **Skalowalność –** Gra musi być stworzona w sposób umożliwiający szybkie i nieskomplikowane dodawanie nowej zawartości, takiej jak poziomy, postacie, elementy graficzne czy mechaniki gry. Struktura projektu powinna umożliwiać modularne rozszerzanie funkcjonalności bez konieczności wprowadzania istotnych zmian w istniejącym kodzie.

3. Stos technologiczny

Tworzenie gry komputerowej to skomplikowany proces. Dlatego aby spełnić założenia funkcjonalne i niefunkcjonalne opisane w podrozdziałach 2.2 oraz 2.3 zdecydowano użyć środowiska Android Studio, które ułatwi implementację na założoną platformę jaką są urządzenia mobilne z systemem Android. Implementacja została napisana w języku programowania Java, który dzięki swojej obiektowej naturze oraz działaniu na maszynie wirtualnej sprzyja stosowaniu dobrych praktyk inżynierii oprogramowania. Obejmują one między innymi enkapsulację, dziedziczenie i polimorfizm. Dodatkowo język Java wymusza podczas implementacji stosowanie podejścia z naciskiem na wydajne wykorzystanie pamięci, wynikające z ograniczonych możliwości bezpośredniego zarządzania nią przez programistę. Powodem jest istnienie systemu *Garbage Collector*, który znosi bezpośredni wymóg zarządzania pamięcią, co jednocześnie upraszcza proces implementacji. W projekcie miejscem przechowywania danych niezbędnych do poprawnego działania gry oraz wznawiania rozgrywki między uruchomieniami jest baza danych SQLite.

3.1. Środowisko programistyczne Android Studio

Android Studio [1] to zintegrowane środowisko programistyczne służące do tworzenia aplikacji pod systemy mobilne Android. Bazuje ono na platformie Intellij IDEA, które same w sobie oferuje potężną podstawę do tworzenia kodu w języku Java. Android Studio wspiera też inne języki programowania takie jak na przykład Kotlin czy C++. Środowisko automatycznie konfiguruje projekt, dostarcza dostęp do bibliotek Android API oraz zarządza procesem kompilacji, co znacząco przyśpiesza tworzenie aplikacji na środowisko Android. Klasy *Canvas* [2] i *SurfaceView* [3] umożliwiają dość intuicyjne i proste tworzenie animacji oraz efektywne renderowanie grafiki 2D w czasie rzeczywistym, co czyni je odpowiednimi narzędziami do implementacji gier typu 2D. Emulator [4] oraz profiler, dostępne w środowisku Android Studio, pozwala analizować wydajność gry. Narzędzia te pozwalają optymalizować zużycie pamięci oraz procesora. Jest jest kluczowe w przypadku aplikacji czasu rzeczywistego, takich jak gry. Domyślna struktura projektu umożliwia wygodne i nieskomplikowane zarządzanie zasobami plikowymi gry. Są to obiekty graficzne, do których należą obrazy sprite'ów oraz czcionki.

3.2. Język programowania Java

Java [5] jest jednym z podstawowych języków programowania obsługiwanych przez środowisko Android Studio i stanowi tradycyjny fundament większości aplikacji tworzonych na ten system. Architektura, elastyczność oraz bogaty ekosystem bibliotek sprawiają, że język ten jest doskonale przystosowany do implementacji gier 2D. Java jest językiem w pełni obiektowym, co oznacza, że logika gry może być zorganizowana w postaci klas i obiektów, takich jak gracz, przeciwnik, pocisk, plansza czy interfejs użytkownika. Dzięki temu kod staje się bardziej czytelny, łatwiejszy w utrzymaniu i rozbudowie. Java w kontekście używania w środowisku Android Studio zapewnia bezpośredni dostęp do interfejsów API platformy Android, dzięki czemu renderowanie grafiki w czasie rzeczywistym jest możliwe.

Java działa pod kontrolą maszyny wirtualnej, co zapewnia wysoki poziom przenośności kodu. Oznacza to, że raz napisana gra może działać na wielu urządzeniach z systemem Android, niezależnie od ich producenta czy wersji systemu. W zależności od punktu spojrzenia pod kątem gier maszyna wirtualna może być wadą lub zaletą w kontekście zarządzania pamięcią. Użytkownik nie zarządza używaną przez program pamięcią dzieje się to przez system *Garbage Collector*. Można interpretować to jako zaletę ze względu na to, że z użytkownika zrzucana jest odpowiedzialność zarządzania pamięcią. Wystąpienie błędów związanych z pamięcią jest dzięki temu minimalizowane. Patrząc z innej perspektywy może to być wada, ponieważ użytkownik przez wyżej opisane działanie nie ma też wpływu kiedy występuje bardziej obciążająca sekwencja instrukcji Garbage Collectora. W kontekście gier, które działają jako nieskończona pętla prowadzić to może to krótkotrwałych przerw w wykonywaniu się instrukcji zawartych w tej pętli co dla klienta, gracza objawi się jako krótkotrwałe zacięcie się gry co jest zjawiskiem, którego należy unikać. Kolejną wadą Garbage Collectora lub zaletą zależnie od interpretacji, jest to że wymusza to na projektancie, programiście pisanie kodu w sposób który unika nadmiernego tworzenia obiektów w pętli gry. Zmusza to do ponownego wykorzystywania istniejących instancji, co jest dobrą praktyką.

3.3. Baza danych SQLite

SQLite [6] to lekka, relacyjna baza danych wbudowana bezpośrednio w system Android. W kontekście środowiska Android Studio i urządzeń mobilnych z systemem Android wszystkie dane zapisywane są w jednym, kompaktowym pliku znajdującym się w pamięci urządzenia. Android SDK natywnie wspiera SQLite co pozwala na bezpośrednie wykorzystywanie tej bazy danych w projektach tworzonych w środowisku Android Studio. SQLite działa w trybie embedded, co oznacza, że baza danych jest uruchamiana w tym samym procesie co aplikacja. Dzięki temu nie ma potrzeby utrzymywania ciągłego połączenia sieciowego ani oddzielnego procesu serwera. Zmniejszone jest też użycie zasobów systemowych w porównaniu z dedykowanymi programami bazodanowymi. Zastosowanie tego rozwiązania pozwoli grze na działanie w trybie offline bez problemów ponieważ wszystkie działania i przechowywanie danych kluczowych odbywa się lokalnie na urządzeniu.

3.4. Program Aseprite

Aseprite [7] jest specjalistycznym narzędziem graficznym przeznaczonym do tworzenia i edycji grafiki pikselowej, powszechnie stosowanej w grach 2D. Program umożliwia projektowanie sprite'ów, tła, ikon oraz animacji w formacie klatkowym (frame-by-frame), co czyni go niezwykle przydatnym podczas tworzenia elementów graficznych do gier. Dzięki intuicyjnemu interfejsowi użytkownika i rozbudowanym funkcjom edycyjnym Aseprite pozwala zachować pełną kontrolę nad każdym pikselem, co jest istotne w kontekście gier o stylistyce retro, *pixelart*. Aseprite wspiera warstwy, przezroczystość, maski i palety kolorów, co znacząco ułatwia tworzenie spójnych, optymalnych pod względem rozmiaru plików graficznych. Narzędzie oferuje również podgląd animacji w czasie rzeczywistym, co umożliwia szybkie testowanie i dopracowywanie ruchów postaci, efektów specjalnych oraz innych dynamicznych elementów gry.

4. Implementacja

Implementacja systemów projektowanej gry odbyła się przy pomocy oferowanych przez środowisko Android Studio bibliotek Java w formie tradycyjnego kodu, z minimalnym wykorzystaniem narzędzi pozwalających na abstrakcję niskopoziomowych operacji. Wszystkie kluczowe elementy gry takie jak główna pętla rozgrywki, mechaniki, interfejsy użytkownika, symulacja obiektów, wzajemne oddziaływanie poszczególnych elementów realizowane jest przy zachowaniu tego podejścia. Podejście to ułatwia późniejszą rozbudowę aplikacji lub zmianę kluczowych mechanik.

4.1. Architektura

Pliki graficzne użyte w zrealizowanej grze zostały zaczerpnięte w kombinacji dwóch źródeł. Strony itch.io [8] na licencji pozwalającej na użytkowanie oraz modyfikowanie zasobów w projektach niekomercyjnych. Oprogramowania Aseprite [7] pozwalającego na tworzenie grafiki 2D. Celem tego podejścia jest zaoszczędzenie czasu na pracy artystycznej co przełoży się na zwiększenie jakości oraz zaawansowania zaimplementowanych systemów.

Przyjęta struktura projektu wzorowana jest na zasadach używanych w paradygmacie obiektowym, który jest popularnym nurtem języka Java. Wszystkie kluczowe elementy zrealizowane zostały w postaci klas oraz ich obiektów. Kombinacja zasad takich jak hermetyzacja, Polimorfizm i dziedziczenie jest niezbędne by zachować czytelną i efektywną implementację założeń aplikacji. Klasy grupowane są w pakietowe przestrzenie nazw, by analiza, rozbudowa i zarządzanie kodem źródłowym było wydajne.

4.2. Pętla Gry

Rozgrywka przebiega w trybie swobodnej gry, bez zdefiniowanych konceptów fabuły lub nadmiernej liniowości. Zawsze aktywna jest tylko jedna instancja rozgrywki, by rozpocząć nową grę trzeba zakończyć obecną, poprzez manualne zakończenie w osobnym menu, lub gdy przeciwnik wygra walkę w bitwie o główną siedzibę gracza. Osadnicy w sami w sobie są jednoznaczni, rozróżnia ich jedynie przydzielony ekwipunek oraz czy przydzielona im rola to wojownik czy pracownik.

4.3. Scena gry

Działania wykonywane w metodach “update” oraz “render” implementacji pętli gry różnią się w zależności od aktywnej sceny gry. Każda ze scen jest osobną klasą nadrzędną posiadającą własną implementację metod interfejsu *GameStateInterface*. W jego skład wchodzą metody renderowania, aktualizacji oraz akcji interakcji użytkownika. Klasy te grupują w sobie obiekty niezbędne do funkcjonowania poszczególnych scen. Wszystkie klasy scen zrealizowane zostały w sposób, który raz przy uruchomieniu aplikacji inicjalizuje je w stanie minimalnym z domyślnymi wartościami danych, a następnie ustawia niezbędne dane w momencie przełączenia się na tę scenę. Podejście to kosztem zwiększonego zużycia pamięci pozwala na zmniejszenie opóźnień między przełączaniem scen co przekłada się na wyższą wydajność.

Przejścia pomiędzy scenami wywoływane są poprzez interakcję użytkownika z obiektem graficznym reprezentującym przycisk w celu zakończenia cyklu oraz przesłania niezbędnych danych do następnej sceny. Listing 4.2. przedstawia metodę interakcji użytkownika dla sceny przygotowawczej przed bitwą. Badane jest miejsce interakcji w koordynatach x i y a następnie porównywane jest z umiejscowieniem graficznych elementów interfejsu użytkownika, jeśli interakcja pokrywa się z umiejscowieniem obiektu, wykonywana jest akcja zgodna z danym obiektem.

Listing 4.1. Funkcja badająca interakcję użytkownika dla sceny przed bitewnej.

```
@Override
public boolean touchEvents(MotionEvent event) {

    if (event.getAction() != MotionEvent.ACTION_DOWN) {
        return true;
    }

    float x = event.getX();
    float y = event.getY();

    if (clickedOnStartButton(x, y)) return true;
    if (clickedOnInventoryMenu(x, y)) return true;
    if (clickedOnUnitStatsMenu(x, y)) return true;
    if (addNewUnitOrSelectExisiting(x, y)) return true;
    if (selectInventorySlot(x, y)) return true;
    if (selectEquipmentSlot(x, y)) return true;
```

```
    return true;
}
```

Listing 4.3. opisuje metodę przejścia z sceny przygotowawczej do sceny symulacji walki. Sprawdzane jest czy użytkownik wszedł w interakcję z obiektem kończącym cykl oraz czy wystawił on drużynę. następnie do sceny walki przekazywane są dane drużyn gracza oraz przeciwnika i zmieniana jest scena. Następna iteracja pętli wykona operacje rysowania i aktualizacji danych przypisanych do sceny symulacji bitwy.

Listing 4.2. Funkcja przełączająca z sceny przed bitewnej na scenę symulacji bitwy.

```
private boolean clickedOnStartButton(float x, float y) {
        if (startButton.getHitbox().contains(x, y)) {
            if (playerTeam.size() == 0) {
                System.out.println("Brak armii");
                return true;
            }
            synchronized (playerTeam) {
                for (Character unit : playerTeam) {
                    unit.setFullHealth();
                }

            }


            synchronized (enemyTeam) {
                for (Character unit : enemyTeam) {
                    unit.setFullHealth();
                }
            }
            Character.setGameState(Game.GameState.BATTLE_PHASE);
            game.setBattlePhase();
            game.setPlayerTeam(playerTeam);
            game.setEnemyTeam(enemyTeam);


            game.passInfoToBattlePhase(InventoryMenu.unitCount);
            game.saveItemsData(InventoryMenu.itemCount);
            InventoryMenu.unitCount = 0;
```

```
            game.setCurrentGameState(Game.GameState.BATTLE_PHASE);
            return true;
        }
        return false;
    }
```

4.4. Wczytywanie grafiki

Wczytywanie grafiki odbywa się przy pomocy typu wyliczeniowego pełniącym funkcję fabryki wizualnych zasobów. Każdy element tego typu reprezentuje inny rodzaj obiektu w grze i posiada własny zestaw obiektów graficznych. W momencie inicjalizacji aplikacji każdy element wczytuje odpowiedni arkusz sprite'ów z pliku graficznego.

Po wczytaniu arkusza sprite'ów wykonywany jest proces jego rozcięcia. Arkusz traktowany jest jako siatka obrazów poziomo ułożone są kolejne klatki animacji, a pionowo różne kierunki animacji. typ wyliczeniowy tworzy tablicę dwuwymiarową, w której każdy wiersz odpowiada jednemu kierunkowi ruchu postaci, a każda kolumna pojedynczej klatce animacji. Z każdego pola siatki wycinany jest osobny fragment grafiki. Dzięki temu animacje jednostek są przygotowane jeszcze przed ich faktycznym użyciem, co skraca czas generowania klatek w trakcie rozgrywki.

Oprócz pełnych animacji tworzone są także ikony używane w interfejsach użytkownika. Pliki graficzne zaprojektowane są w sposób, który pozwala systemowi na proste załadowanie bitmap przeznaczonych do użycia w interfejsie użytkownika, np. w panelach wyboru jednostki. W przeciwieństwie do podstawowych klatek animacji, ikona może być później skalowana dynamicznie na żądanie. Możliwe też jest pobranie konkretnych klatek na podstawie współrzędnych tablicy oraz skalowanie wybranych elementów o podany współczynnik.

Dzięki takiej architekturze cały system animacji jednostek jest wydajny, ponieważ wszystkie kosztowne operacje graficzne takie jak wczytywanie z plików, rozcinanie czy skalowanie wykonywane są tylko raz, podczas inicjalizacji programu. Późniejsze pobieranie i użytkowanie zasobów polega jedynie na zwróceniu bitmapy z przygotowanej tablicy.

4.5. System przedmiotów

Podobnie jak w przypadku pozostałych systemów gry, zrezygnowano z narzędzi generujących wysokopoziomowe abstrakcje, stawiając na pełną kontrolę nad strukturą danych oraz przepływem logiki. Zastosowanie typów wyliczeniowych tak jak w systemie wczytywania grafiki w rozdziale 4.4 pozwala na zdefiniowanie wszystkich parametrów przedmiotów w sposób statyczny, rozproszony i bezpieczny typowo, co ułatwia korzystanie z nich w logice gry oraz minimalizuje ryzyko błędów.

Kluczowym elementem architektury jest nadrzędny typ wyliczeniowy Przedmioty, który pełni rolę grupującego katalogu dla wszystkich typów przedmiotów dostępnych w grze. Każda kategoria zawiera własną tablicę typów, np. broń, tarcza, zbroja, zasób metalu, które reprezentują konkretne warianty danego przedmiotu. Wszystkie typy przedmiotów implementują wspólny interfejs, który definiuje zestaw właściwości wymaganych dla mechanik gry. Należą do nich między innymi:

- współrzędne graficzne (x, y) wykorzystywane przez system renderowania, by znaleźć pozycję graficzną w tablicy wczytanych bitmap,
- indeks kategorii, na którego podstawie identyfikowane są funkcje danego przedmiotu, przykładowo hełm należy do indexu określającego możliwość wyposażenia w niego jednostki, a sztabka miedzi do indeksu określającego możliwość użycia jej w systemie wytwarzania,
- tablica liczb typu prymitywnego zawierającego zestaw liczb określających statystyki przedmiotu, mająca znaczenie w przypadku jeśli dany przedmiot może został wyekwipowany przykładową statystyką zawartą w tej strukturze danych jest liczba określająca punkty zdrowia, które uzyska jednostka wyposażona w dany przedmiot,
- osobna tablica liczb która gromadzi dane odnośnie kosztów i czasu potrzebnego do stworzenia danego przedmiotu w systemie wytwarzania.

Zaprojektowana w ten sposób struktura przedmiotów zapewnia, że wszystko od miecza, przez hełm, po sztabkę metalu jest obsługiwane przez jednolity zestaw metod, co upraszcza integrację z pozostałymi systemami gry, takimi jak wytwarzanie, ekwipunek czy

system bitewny. Na listingu 4.4 przedstawione jest definiowanie nowych typów przetworzonego materiału metalu. Dzięki tej strukturze dodanie nowego przedmiotu to kwestia wpisania go do odpowiedniego typu wyliczeniowego z danymi go definiującymi co pozwala na bezproblemowe implementowanie i rozszerzanie oferowanych w grze rzeczy.

Listing 4.3. Struktura przedstawiająca sposób implementacji specjalnych materiałów.

```
enum ResourcesIngotsType implements SpriteType {
        BRONZE_INGOT(12,0,5,1, 1,1),

        IRON_INGOT(12,2,5,25, 5,25),

        GOLD_INGOT(12,3,5,50, 5,45),

        DIAMOND_INGOT(12,4,5,100, 5,70),

        MAGMA_INGOT(12,5,5,200, 5,120);
private final int x, y, index,goldCost,materialCost,craftingTime;

        ResourcesIngotsType( int y, int x, int index, int goldCost,
int materialCost, int craftingTime) {
            this.x = x;
            this.y = y;
            this.index = index;
            this.goldCost = goldCost;
            this.materialCost = materialCost;
            this.craftingTime = craftingTime;
        }
    }
```

Listing 4.4. Struktura przedstawiająca sposób implementacji przedmiotów ekwipunku.

```
enum WeaponType implements SpriteType {
        BRONZE_SWORD(20, 14,0, WEAPON,
     new int[]{GameConstants.Stats.DAMAGE,
    GameConstants.Stats.WEAPON_TYPE},
     new int[]{50,GameConstants.WeaponTypes.SWORD}, 1, 1, 1),

            BASIC_BOW(4, 4, 0, WEAPON,
     new int[]{GameConstants.Stats.DAMAGE,
    GameConstants.Stats.WEAPON_TYPE}, new
```

```
int[]{30,GameConstants.WeaponTypes.BOW}, 1, 1, 3),

        GOLD_SWORD(21, 1, 0, WEAPON,
 new int[]{GameConstants.Stats.DAMAGE,
GameConstants.Stats.WEAPON_TYPE}, new
int[]{150,GameConstants.WeaponTypes.SWORD}, 100, 8, 120),

        DIAMOND_SWORD(20, 15, 0, WEAPON,
 new int[]{GameConstants.Stats.DAMAGE,
GameConstants.Stats.WEAPON_TYPE}, new
int[]{225,GameConstants.WeaponTypes.SWORD}, 200, 10, 240),

        MAGMA_SWORD(21, 2, 0, WEAPON,
 new int[]{GameConstants.Stats.DAMAGE ,
GameConstants.Stats.WEAPON_TYPE}, new
int[]{350,GameConstants.WeaponTypes.SWORD}, 350, 15, 360);

        private final int x,
y,index,goldCost,materialCost,craftingTime;
        private final int[] affectedStats, stats;
        private final ItemCategory parentCategory;

        WeaponType(int y, int x, int index, ItemCategory
parentCategory, int[] affectedStats, int[] stats, int goldCost,
int materialCost, int craftingTime) {
            this.x = x;
            this.y = y;
            this.index = index;
            this.parentCategory = parentCategory;
            this.stats = stats;
            this.affectedStats = affectedStats;
            this.goldCost = goldCost;
            this.materialCost = materialCost;
            this.craftingTime = craftingTime;}
        }
```

4.6. System zapisywania danych rozgrywki

System zapisywania i ładowania danych w grze wykorzystuje wbudowane mechanizmy relacyjnej bazy danych SQLite dostępne w środowisku Android Studio. Podobnie jak w pozostałych elementach projektu, zastosowano tradycyjne podejście oparte na ręcznie definiowanych zapytaniach SQL, co daje pełną kontrolę nad strukturą tabel oraz

zachowaniem bazy podczas uruchamiania gry. Cała logika obsługi bazy danych została umieszczona w dedykowanej klasie, która odpowiada za:

- inicjalizację pliku bazy danych,
- tworzenie tabel wykorzystywanych przez systemy gry,
- wypełnianie tabel wartościami domyślnymi jeśli dochodzi do rozpoczęcia nowej rozgrywki

Takie podejście pozwala na oddzielenie logiki zarządzania danymi od pozostałych systemów gry i daje gwarancję, że struktura danych zawsze jest poprawnie zainicjalizowana w momencie pierwszego uruchomienia aplikacji. Struktura każdej tabeli została przygotowana ręcznie za pomocą komend SQL. Każda tabela posiada unikalny identyfikator oraz wartości dopasowane do logiki systemów gry.

- Tabela zasobów zawiera wpis z wierszami opisującymi jakie i ile danych zasobów posiada gracz oraz ich górne limity definiowane przez budynki typu magazynu znajdują się w niej wpisy takie jak ilość drewna, kamienia, złota czy populacja,
- Tabela przedmiotów zawiera wpis z wierszami opisującymi jakie i ile przedmiotów posiada w swoim magazynie gracz,
- Tabela budynków zapisująca informacje dotyczące danych niezbędnych dla rozmieszczenia budynków i ich typu.

Dane gromadzone w bazie danych służą do odczytywania stanu rozgrywki podczas zmiany sceny lub ponownym otworzeniu aplikacji.

5. Interfejs użytkownika

Każda ze scen posiada własny interfejs pozwalający na zarządzanie zasobami materialnymi oraz jednostkami. Głównymi elementami są przyciski pozwalające na otworzenie menu lub wyświetlenie siatki z którą interakcja pozwala na przykład na postawienie nowego budynku w obozie.

5.1. Scena zarządzania obozem

Interfejs użytkownika w tej scenie, przedstawiony na rysunku 5.1 podzielony jest na trzy grupy. Środkowa część interfejsu przedstawia główny ekran sceny wyświetla on obecny stan obozu gracza oraz zasoby naturalne do których ma dostęp użytkownik. Górna część ekranu przedstawia panel, który w strukturze ikona - liczba przedstawia zaopatrzenie gracza w surowce. Dolny panel zawiera przyciski pozwalające między innymi na zakończenie obecnego cyklu.

Rys. 5.1. Interfejs sceny obozu [Opracowanie własne]

5.1.1 Panel zasobów

Zadaniem górnego panelu jest wizualne przedstawienie zasobów dostępnych dla gracza. Nie zawiera on wszystkich możliwych w grze materiałów, ale tylko te najczęściej używane. Zaopatrzeniem zawsze wyświetlanym w tym panelu są materiały fundamentalne takie jak złoto. Drewno oraz kamień pozyskiwane ze środowiska przy pomocy odpowiednich budynków są główną walutą potrzebną do konstrukcji nowych budynków. Materiały następne to zasoby “przemysłowe”, bez nich niemożliwym jest wytwarzanie przedmiotów, w które wyekwipować można załogę gracza podczas bitew. Z prawej strony znajduje się populacja osady. W formacie liczba / liczba użytkownikowi przedstawiany jest górny limit jaki może zaspokoić obecna osada oraz liczbę przydzielonych pracowników limit ten zwiększa się budując nowe domy w osadzie.

5.1.2 Panel przycisków

Funkcją tego panelu jest zgrupowanie w jednym miejscu przycisków pozwalających na wykonywanie bardziej ogólnych zadań. Czynności takie jak wyświetlanie menu zawierającego dokładniejsze przedstawienie magazynów, zakończenie cyklu zarządzania obozem oraz panel pozwalający na wydanie rozkazu konstrukcji nowego budynku wywoływane są z tego panelu po kliknięciu odpowiedniego przycisku. W rogu znajduje się przycisk, który wyświetla menu w jakim możliwym jest zresetowanie rozgrywki do punktu startowego i zapisanie liczby cykli jakie gracz osiągnął w porzuconej rozgrywce. Menu wyświetla też 5 wyników z największą minioną liczbą cykli.

Rys. 5.2. Panel przycisków [Opracowanie własne]

5.1.3 Menu rozbudowy osady

Jest to menu wyświetlające jakie budynki zbudować może gracz oraz wiążące się z tym koszty. panel ten wyświetlany jest po kliknięciu przycisku z ikoną planu budowlanego i młotka na panelu przycisków. Po zdecydowaniu się na budowę i wyborze struktury z listy menu zamyka się i i wyświetlana jest siatka, której zadaniem jest wizualne przedstawienie miejsc w których możliwe jest postawienie nowego budynku.

Rys. 5.3. Interfejs rozbudowy, menu wyboru budowy [Opracowanie własne]

Rys. 5.4. Interfejs rozbudowy, siatka wyboru miejsca budowy [Opracowanie własne]

5.1.4 Menu szczegółowe budynku

Każdy wybudowany budynek ma dedykowane menu pozwalające na wykonywanie zadań związanymi z tymi budynkami, przykładowo kuźnia pozwala na dostęp do interfejsu wytwarzania a tym samym na interakcję z tym systemem. Każdy budynek funkcyjny umożliwia przypisanie osadnika. Populacja spełnia dwie role, robotnicy oraz żołnierze. Gra umożliwia użytkownikowi przydzielenie mieszkańców do budynków, by mogli zasilić magazyn zasobów oraz wytworzyć ekwipunek.

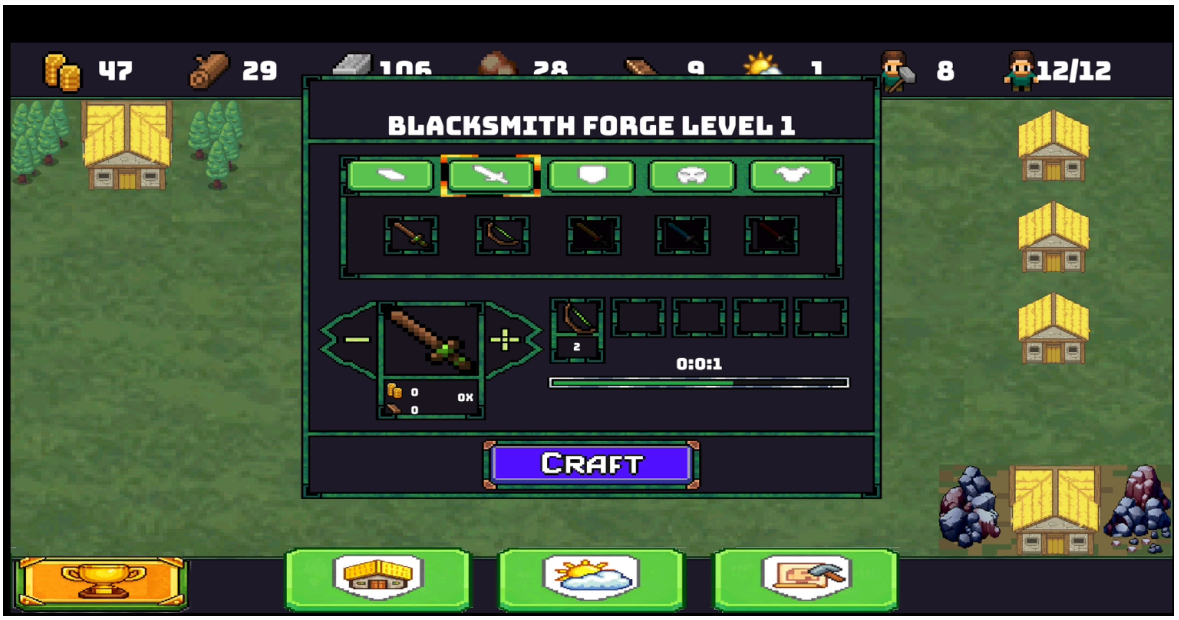
Rys. 5.5. Interfejs budynku [Opracowanie własne]

5.1.5 Menu wytwarzania zasobów

Proces przetwarzania wydobytych zasobów oraz wytwarzanie elementów ekwipunku odbywa się przy pomocy panelu dostępnego jako menu funkcyjne budynku kuźni. Dostęp do tego panelu wymaga uprzedniego wybudowania samej kuźni, następnie po zakończeniu budowy należy wejść w interakcję z obiektem, by następnie pokazane zostało menu wytwarzania. Interfejs podzielony jest na 3 główne segmenty:

1. Segment katalogowy - pozwala na przełączenie pomiędzy kategoriami przedmiotów oraz wyświetlenie jakie przedmioty z danej kategorii są możliwe przygotowania.

2. Segment wyboru - po wybraniu przedmiotu z okna katalogowego wybór ten wyświetlany jest w tym oknie wraz z szczegółami dotyczącymi kosztów produkcji ikony “+” oraz “-” pozwalając na zmianę ilości w zleceniu produkcji.
3. Segment kolejki - przedstawia listę produkcji a poniżej pasek odliczający czas potrzebny na wyprodukowanie jednego przedmiotu z pierwszej pozycji list.



Rys. 5.6. Interfejs wytwarzania przedmiotów [Opracowanie własne]

5.2. Scena przygotowawcza przed bitwą

Scena, przedstawiona na rysunkach 5.7 przedstawia siatkę z możliwymi pozycjami dla jednostek gracza oraz przeciwnika. Interfejs użytkownika zaimplementowany został w sposób gdzie menu ekwipunku jednostek oraz zaopatrzenia są ukryte do momentu gdy gracz nie wejdzie w interakcję z wystawioną jednostką lub w przypadku menu zaopatrzenia nie kliknie odpowiedniego przycisku.

5.2.1 Menu zaopatrzenia

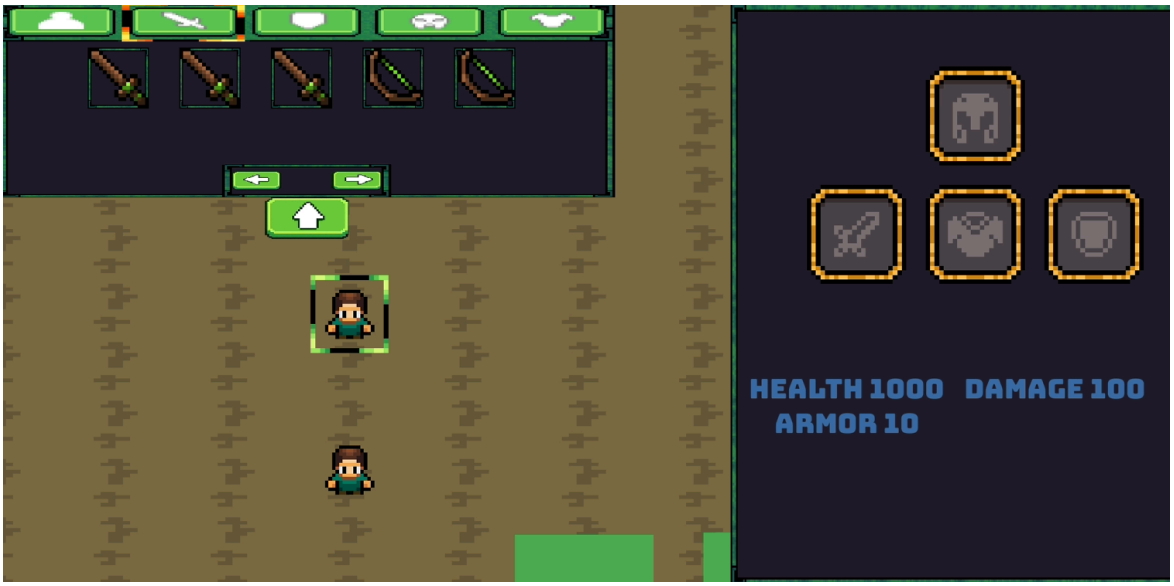
Po kliknięciu odpowiedniego przycisku rozwinięte zostanie menu składające się z panelu kategorii, które filtruje wyświetlane zaopatrzenie filtry zawierają między innymi dostępnych osadników, broń, przedmioty drugiej ręki, nakrycia głowy czy zbroje. Poniżej wyświetlany jest aktualny stan zaopatrzenia, gracz może wejść w interakcję z obiektem z

listy. W przypadku wyboru osadnika wyświetlona zostanie siatka z możliwymi miejscami, w których wystawiona może zostać jednostka.

Rys. 5.7. Interfejs zaopatrzenia [Opracowanie własne]

5.2.2 Menu wyposażenia jednostki

Gdy gracz wejdzie w interakcję z przedmiotem zostanie on podświetlony, ale nie stanie się nic póki użytkownik nie wybierze wystawionej już jednostki. Po tej interakcji wyświetlone zostanie menu ekwipunku zaznaczonego wojownika tak jak zaprezentowane jest to na rysunku 5.8. Menu to zawiera informację o możliwych do wyposażenia przez jednostkę typach przedmiotów, jakie przedmioty obecnie posiada oraz aktualne statystyki tejże jednostki. Interakcja polega na kliknięciu w odpowiednią ikonę w wyświetlonym interfejsie. W przypadku wybrania jednego z tych pól system weryfikuje stan widoczności okna zaopatrzenia. W przypadku, w którym nie jest ono w stanie wyświetlania, automatycznie zostaje otwarte. Po potwierdzeniu widoczności system wybiera kategorię zgodną z wybranym przez użytkownika polem ekwipunku wojownika. Rozwiązanie to ogranicza pulę widocznych przedmiotów do kompatybilnych z wybranym miejscem wyposażenia. Kliknięcie w przedmiot wyekwipuje jednostkę w ten przedmiot aktualizując statystyki wojownika o statystyki wybranego przedmiotu.



Rys. 5.8. Interfejs wyposażenia danej jednostki [Opracowanie własne]

Jeśli gracz uzna ustawienie armii oraz jej wyposażenie za odpowiednie to akceptuje stworzone rozstawienie i płynnie przechodzi do sceny walki. Wystawiona armia gracza i przeciwnika symuluje walkę, przykład widnieje na rysunku 5.9. Po jej zakończeniu graczowi zostaje przedstawione menu podsumowujące z którego następnie występuje przeniesienie z powrotem na ekran zarządzania obozem.

Rys. 5.9. Przykładowa symulacja walki [Opracowanie własne]

6. Testy

Aplikacja testowana była w sposób manualny, w celu sprawdzenia i potwierdzenia poprawnej działalności kluczowych mechanik gry oraz czy są zgodne z założeniami projektu. Testy obejmują między innymi sprawdzenie interakcji użytkownika z osadą, wystawienie drużyny do walki oraz jej wyposażenie. Testy opisane zostały w formie instrukcji jaką musi wykonać tester wraz z oczekiwanym zachowaniem aplikacji.

6.1. Testy zarządzania obozem

Test 001 (Tabela 6.1.) ma na celu potwierdzenie poprawności działania systemu określającego pozycję budynków w obozie oraz przypisania pracowników. Kliknięcie w sprite budynku wyświetla menu informacyjne zgodne z realnym stanem wizualnym wybranej konstrukcji oraz jej kategorią. Menu to posiada również liczbę przypisanych pracowników.

Rys. 6.1. Menu budynku głównego [Opracowanie własne]

Tabela 6.1. Scenariusz testowy - interakcja z budynkiem

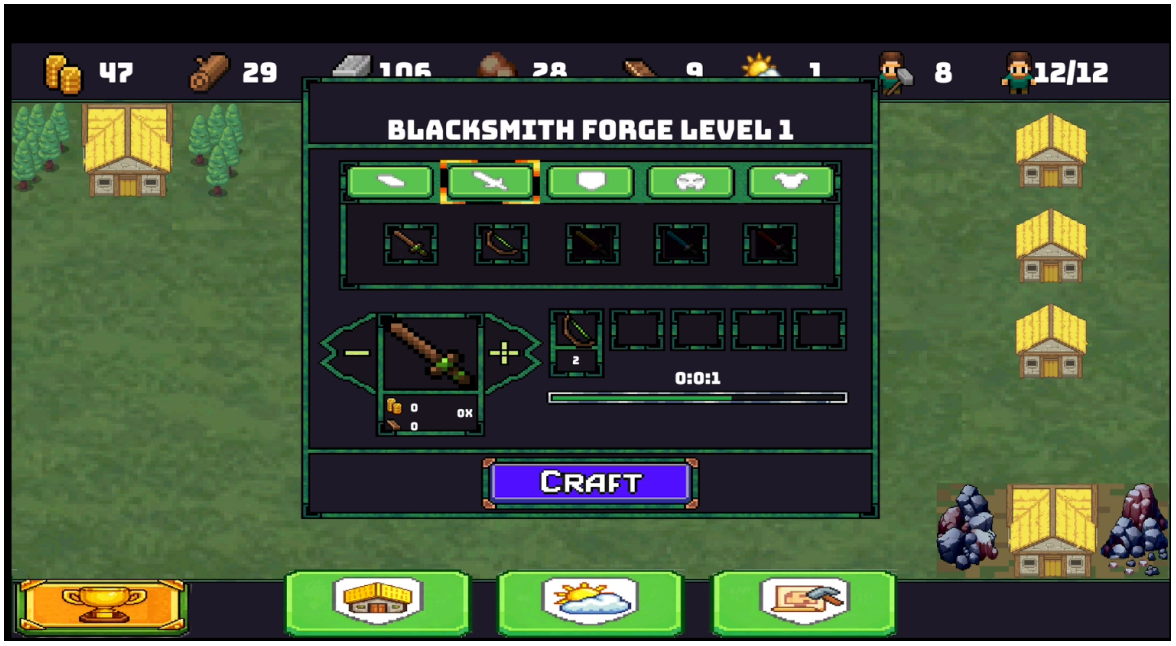
ID	001
Tytuł	Interakcja z budynkiem oraz przydzielanie pracowników
Warunki początkowe	Pierwsze uruchomienie gry / Powrót do sceny obozu z sceny bitewnej
Kroki testowe	1. Zlokalizowanie głównego budynku 2. Kliknięcie go myszą w przypadku emulatora lub palcem w przypadku urządzenia mobilnego 3. Kliknięcie w ikonę “+” lub “-” by zmienić liczbę przypisanych pracowników
Oczekiwany rezultat	Użytkownikowi powinno ukazać się menu budynku z którym zaszła interakcja, a interakcja kolejno z ikoną “+” zwiększy liczbę przypisanych pracowników gdy “-” zmniejszy. W obu przypadkach wartość nie powinna przekroczyć limitów

Test 002 (Tabela 6.2) ma na celu potwierdzenie poprawności działania systemów rozbudowy obozu. Jeśli gracz wybierze budynek z menu rozbudowy, ale nie spełnia warunków potrzebnych do rozpoczęcia konstrukcji wybranej struktury takich jak potrzebne materiały wtedy akcja zostanie odrzucona przez system. Po spełnieniu pierwszego warunku sprawdzane jest, czy wybrane przez gracza miejsce budowy jest dozwolone, w przypadku pozytywnym system zainicjuje budowę.

Tabela 6.2. Scenariusz testowy - interakcja z menu rozbudowy

ID	002
Tytuł	Interakcja z menu rozbudowy
Warunki początkowe	Pierwsze uruchomienie gry / powrót do sceny obozu z sceny bitewnej
Kroki testowe	1. Kliknięcie na przycisk rozbudowy z dolnego menu 2. Wybranie budynku z wyświetlonego menu 3. Kliknięcie na jedno z możliwych miejsc w obozie
Oczekiwany rezultat	Użytkownik po kliknięciu w przycisk rozbudowy wyzwoli menu zawierające listę możliwych do wybudowania budynków. Wybranie jednego z nich wyświetli siatkę migającą na 2 kolory, czerwony oznaczający miejsce niemożliwe do postawienia nowego budynku oraz niebieski, który oznacza miejsce w którym budowa jest możliwa. Wybranie miejsca z niebieskim kolorem powinno zainicjalizować konstrukcję budynku

Test 003 (Tabela 6.3.) ma na celu potwierdzenie poprawności działania systemów odpowiedzialnych za przetwarzanie materiałów. Gracz po zbudowaniu miejsca przetwarzania i wejściu z nim w interakcję będzie miał dostęp do menu wytwarzania, gdzie przy założeniu, że spełnia warunki kosztów będzie mógł wytworzyć wybrany przedmiot.



Rys. 6.2. Menu wytwarzania (Opracowanie własne)

Tabela 6.3. Scenariusz testowy - interakcja z systemem wytwarzania

ID	003
Tytuł	Interakcja z systemem wytwarzania i przetwarzania przedmiotów
Warunki początkowe	Pierwsze uruchomienie gry / Powrót do sceny obozu z sceny bitewnej
Kroki testowe	1. Jeśli nie wybudowano odpowiedniego budynku powtórzyć instrukcję opisane w teście 002 dla budynku kategorii “kuźnia” 2. Powtórzenie kroków testu 001 w celu wyświetlenia menu budynku “kuźnia” 3. Kliknięcie przycisku wytwarzania w ukazanym menu 4. Wybranie przedmiotu wytworzenia 5. Zatwierdzenie wyboru

Oczekiwany rezultat	Użytkownik po wejściu w interakcję z kuźnią dostaje dostęp do menu umożliwiającego przetwarzanie zasobów i wytwarzanie ekwipunku. Wybranie przedmiotu z listy oraz kliknięcie w przycisk zatwierdzenia kolejkuje ten przedmiot i po upływie określonego czasu zostanie on dodany do magazynku gracza

6.2. Testy mechanik bitwy

Użytkownik po zakończeniu etapu gry związanego z zarządzaniem bazą by przejść dalej musi kliknąć w przycisk znajdujący się u dołu ekranu, którego funkcją jest zatwierdzenie zakończenia tego etapu rozgrywki. Graczowi ukaże się mapa terenu wokół obozu, na której występować mogą oddziały wroga, do gracza należała będzie decyzja czy chce zakończyć cykl czy zainicjować potyczkę z wybraną przez siebie grupą wroga. W przypadku podjęcia walki gracz przeniesiony zostanie na ekran bitwy gdzie ma dostęp do rozplanowania swojej drużyny oraz wybrania jakiego ekwipunku będzie używać.

Rys. 6.3. Scena nad obozem [opracowanie własne]

Rys. 6.4. Rozwinięte menu zaopatrzenia [opracowanie własne]

Test 004 (Tabela 6.4.) ma na celu potwierdzenie poprawności działania menu zaopatrzenia. Kliknięcie w przycisk w górnej części ekranu wyświetla menu informacyjne zawierające stan magazynu zgodny z realnym stanem na moment zakończenia etapu zarządzania obozem.

Tabela 6.4. Scenariusz testowy - interakcja z menu zaopatrzenia

ID	004
Tytuł	Interakcja z menu zaopatrzenia
Warunki początkowe	Zakończenie cyklu zarządzania bazą oraz wybranie przeciwnika
Kroki testowe	1. Kliknięcie przycisku zakończenia cyklu zarządzania obozem 2. Wskazanie z mapy podsumowania celu do ataku (jeśli takowy istnieje) 3. Po przejściu na scenę przedbitewną kliknięcie przycisku rozwijającego menu zaopatrzenia 4. Kliknięcie każdego z przycisków

	kategorii zaopatrzenia 5. Kliknięcie poza menu zaopatrzenia lub ponownie w przycisk, który za pierwszym razem ujawnił menu
Oczekiwany rezultat	Użytkownik po wykonaniu kroków 1 oraz 2 zostanie przeniesiony na pole bitwy gdzie widoczna będzie już wystawiona drużyna przeciwnika. W górnej części ekranu znajduje się przycisk a kliknięcie w niego ukaże menu zaopatrzenia. kliknięcie w każdy z przycisków będących częścią wyświetlonego menu filtruje wyświetlane zaopatrzenie do określonej kategorii. Kliknięcie poza teren interfejsu zainicjuje zwinięcie i ukrycie go

Test 005 (Tabela 6.5.) ma na celu potwierdzenie poprawności działania systemu wystawiania drużyny gracza. W tym celu po otwarciu menu zaopatrzenia i wybraniu ikony osadnika wyświetli się siatka wizualizująca możliwe pozycje na polu bitwy po wybraniu jednej z nich na polu bitwy w wybranym miejscu pojawi się wojownik.

Rys. 6.5. Wystawienie wojownika [opracowanie własne]

Tabela 6.5. Scenariusz testowy - wybranie drużyny mającej wziąć udział w bitwie

ID	005
Tytuł	Wystawienie drużyny
Warunki początkowe	Zakończenie cyklu zarządzania bazą oraz wybranie przeciwnika
Kroki testowe	1. Kliknięcie przycisku zakończenia cyklu zarządzania obozem 2. Wskazanie z mapy podsumowania celu do ataku (jeśli takowy istnieje) 3. Po przejściu na scenę przedbitewną kliknięcie przycisku rozwijającego menu zaopatrzenia 4. Kliknięcie w przycisk kategorii osadników 5. Wybranie osadnika z listy 6. Wybranie jednego z kafelków na polu bitwy
Oczekiwany rezultat	Użytkownikowi po wybraniu osadnika z listy menu zaopatrzenia wyświetli się siatka możliwych pozycji wystawienia jednostki. Kliknięcie jednego z miejsc zatwierdzi wybór, wystawi wojownika w tym miejscu oraz usunie go z listy zaopatrzenia

Test 006 (Tabela 6.6.) ma na celu potwierdzenie poprawności działania systemu wystawiania drużyny gracza. W tym celu po otwarciu menu zaopatrzenia oraz wybraniu ikony osadnika, wyświetli się siatka przedstawiająca dostępne pozycje rozmieszczenia jednostek na polu bitwy. Następnie, po wskazaniu jednej z dostępnych pozycji, w wybranym miejscu zostaje umieszczony wojownik.

Rys. 6.6. Wyposażenie wojownika w przedmiot [opracowanie własne]

Tabela 6.6. Scenariusz testowy - wyposażenie wojownika w ekwipunek

ID	006
Tytuł	wyposażenie wojownika
Warunki początkowe	wystawienie wojownika na polu bitwy
Kroki testowe	1. Kliknięcie wojownika 2. wybranie pola ekwipunku 3. wybranie przedmiotu zaopatrzenia
Oczekiwany rezultat	Użytkownikowi po wybraniu wojownika ukażę się jego ekwipunek. Wybranie pola wyposażenia automatycznie wyświetli menu zaopatrzenia wraz z odpowiednim filtrem. Wybranie przedmiotu usunie ten przedmiot z zaopatrzenia i przeniesie go do ekwipunku wojownika.

7. Podsumowanie i wnioski

Celem niniejszej pracy, było zaprojektowanie oraz implementacja gry strategicznej 2D, która łączy w sobie elementy zarządzania zasobami ludzkimi oraz zaopatrzeniem. Interfejs użytkownika został zaprojektowany w sposób intuicyjny i spójny z klimatem gry. Użytkownik może wytwarzać ekwipunek używając wydobytych w obozie materiałów oraz wykorzystywać go w prowadzonych potyczkach poprzez przydzielanie go poszczególnym jednostkom. Udało się zaimplementować prostolinijną pętlę rozgrywki spełniającą założenia projektu. Praca spełnia założone wymagania funkcjonalne i niefunkcjonalne. Proces projektowy uwzględniał zarówno aspekty wizualne, jak i techniczne, a głównym założeniem było stworzenie środowiska gry, które pozwoli graczowi na podejmowanie decyzji w sprawie dostępnych mu zasobów.

Grafiki wykorzystane w projekcie, bazują na zasobach dostępnych na licencji Open-Source, co pozwoliło na zachowanie wysokiej jakości wizualnej, przy minimalnych kosztach. Część grafik wykonana została własnoręcznie, aby dopasować elementy do szaty graficznej występującej w grze.

Gra posiada potencjał do dalszego rozwoju i usprawnień. Podział każdego elementu gry na osobne komponenty oraz sposób implementacji kodu odpowiedzialnego za przedmioty oraz istoty występujące w grze pozwala na dodanie nowych typów oraz rodzajów tych obiektów, bez konieczności ingerencji w istniejącą logikę zarządzającą systemami. Gra może zostać rozbudowana o dodatkowe funkcje, takie jak: rozbudowanie systemu przedmiotów o obsługiwanie bardziej skomplikowanych typów broni, przykładowo kosturów pozwalających użytkownikowi na wykonywanie ataków magicznych. Dodanie systemów pozwalających przeciwnikowi na podejmowanie bardziej skomplikowanych decyzji niż prostolinijne nacieranie na siedzibę gracza. W grze można również rozbudować elementy wizualne oraz animacje na przykład poprawienie przejść między scenami czy dodanie nowych i bardziej dopracowanych assetów graficznych.

Podsumowując, zrealizowana aplikacja spełnia wszystkie założone wymagania. Realizacja projektu przyczyniła się do znaczącego poszerzenia wiedzy w zakresie projektowania, implementacji gier oraz optymalizacji wydajności. Zdobyte doświadczenie stanowi solidną podstawę do realizacji kolejnych projektów w obszarze tworzenia gier.

Bibliografia

[1] Neil Smyth. Android Studio 3.5 Development Essentials - Java Edition. Developing Android 10 (Q) Apps Using Android Studio 3.5, Java, and Android Jetpack

[2] Herbert Schildt. Java. Kompendium programisty. Wydanie VIII, wyd. Helion

[3] Środowisko Android Studio
https://developer.android.com/studio/intro?utm_source=chatgpt.com&hl=pl

[4] Android Studio klasa Canvas

https://developer.android.com/reference/android/graphics/Canvas

[5] Android Studio klasa SurfaceView

https://developer.android.com/reference/android/view/SurfaceView

[6] Android Studio emulator

https://developer.android.com/studio/run/emulator?hl=pl

[7] Oracle Java Documentation

https://docs.oracle.com/en/java/javase/25/

[8] Android Studio SQLite

https://developer.android.com/training/data-storage/sqlite?hl=pl

[9] Program Aseprite

https://www.aseprite.org/

[10] Strona Itch.io

https://itch.io/

Spis rysunków

Spis tabel

Spis listingów