Imię i nazwisko:
Filip Żurek
Grupa (dzień, godzina):
Wtorek, 15:00

1. Temat aplikacji (*np. gra detektywistyczna, symulator lotu na Marsa):*
Symulator turysty (ang. Tourist Simulator) / „Alone Tourist" – gra RPG, w której gracz wciela się w turystę zwiedzającego miasto i jego atrakcje.

2. Czynności wykonywane przez użytkownika pod względem MERYTORYCZNYM (*np. podejmowanie decyzji i rozwiązywanie zagadek, określanie parametrów lotu*):
Gracz będzie mógł podjąć decyzję o zwiedzeniu zabytków, kupnie pamiątek, jedzenia itp. Turysta będzie musiał zarządzać swoim czasem i budżetem, na początku rozgrywki gracz ustala, ile ma pieniędzy na wyjazd oraz o której godzinie wstaje (rozpoczyna dzień zwiedzania).

3. Czynności wykonywane przez użytkownika pod względem TECHNICZNYM (*np. klikanie przycisków na klawiaturze, dostarczanie plików z parametrami wejściowymi)*:
Gracz będzie poruszał się po mapie w stylu gry „A dark room" przy wykorzystaniu klawiszy „WSAD" lub strzałek. Dodatkowo będzie podejmował decyzję o zwiedzaniu zabytku, na który trafi. Przy wybraniu opcji zwiedzania będzie mógł podjąć dodatkowe interakcje w obiektach w stylu: zrób zdjęcie, kup pamiątkę co dodatkowo wpłynie na parametry rozgrywki.

4. Oczekiwane rezultaty (*np. gra kończy się prawidłowym lub nieprawidłowym wskazaniem mordercy; symulacja zwraca informację o tym, czy udało się dolecieć na Marsa)*:
Gracz zwiedza zabytki, eksploruje miasto a gra kończy się, kiedy gracz podejmie decyzję o jej zakończeniu przed czasem lub po powrocie do hotelu na zakończenie i podsumowanie dnia. Ewentualnym zakończeniem jest wyczerpanie zasobów czasu / pieniędzy (wyzerowanie portfele nie oznacza zakończenia jednak ograniczenie funkcjonalności).

5. Proszę zidentyfikować jeden przypadek wykorzystania polimorficznej metody, który na pewno trzeba będzie umieścić w programie *(np. Metoda [nazwa] w klasie/interfejsie [nazwa] i klasach dziedziczących – w zależności od klasy, metoda ta będzie...)*:
W programie trzeba umieścić metodę [Visit()] w interfejsie [IAtraction()] zaimplementowaną w klasach: Museum(), Park(), Castle(), Restaurant(). W każdej klasie metoda Visit() będzie realizować

inną logikę zależnie od typu atrakcji. Przykładowo, żeby odwiedzić muzeum trzeba kupić bilet (co wpłynie na stan portfela), w trakcie wizyty będzie można podjąć decyzję o zrobieniu zdjęcia czegoś (+ x czas) lub zakupie pamiątek (+ x czas; - y pieniądze). Innym typem atrakcji będzie park lub restauracja, gdzie wstęp jest darmowy, ale opcje do wyboru będą inne i odpowiednio będą oddziaływać na statystyki rozgrywki.

6. Proszę zidentyfikować jakiś rodzaj relacji między klasami inny niż dziedziczenie wykorzystywane w punkcie 5, który trzeba będzie umieścić w programie *(np. Klasa [nazwa1] połączona z klasą [nazwa2] przy pomocy dziedziczenia/agregacji/kompozycji/zależności...)*:

Klasa Tourist() będzie połączona przy pomocy kompozycji z klasą Inventory(), ponieważ turysta będzie posiadał swój ekwipunek, w którym będzie przechowywał zakupione pamiątki, zrobione zdjęcia, pieniądze itp. Wykorzystanie do tego celu kompozycji wydaje się sensowne z tego powodu, że bez turysty jego ekwipunek nie istnieje.

7. Proszę zidentyfikować i nazwać trzy klasy, które nie zostały wymienione w punktach 5 i 6, a które na pewno trzeba będzie umieścić w programie *(np. Klasy [nazwa1], [nazwa2], [nazwa3])*:

Map() – klasa odpowiedzialna za przechowywanie układu miasta i atrakcji;

TimeManager() – odpowiada za kontrolowanie czasu dnia w grze;

Souvenir() – klasa reprezentująca pojedynczy przedmiot, który można zakupić w sklepach z pamiątkami i atrakcjach, przechowywany później w Inventory();

Game() – główna klasa sterująca gry;

8. Miejsce na ewentualne uwagi lub pytania do prowadzącego:

Grę można następnie rozbudować o mechaniki odpowiadające za utrzymywanie postaci przy życiu, czyli kontrolowanie przez gracza poziomu nawodnienia, pożywienia. Dodatkowym pomysłem byłoby wdrożenie poziomu wyczerpania, który spadałby w ciągu dnia, jego poziom można byłoby zwiększyć przez odpoczynek, wyczerpanie poziomu zdrowia prowadziłoby do omdlenia. Dalszym sposobem na przedłużenie rozgrywki byłoby wprowadzenie trwania gry przez parę dni (tur: „cały wyjazd") a nie tylko jeden dzień z jego podsumowaniem.

W kodzie programu możliwe jest zaimplementowanie wzorców projektowych takich jak: Strategia (do zarządzania różnymi zachowaniami atrakcji), Fabryka (tworzenie atrakcji: Museum, Park, Castle, Restaurant), Decorator (upiększanie atrakcji np. Guided Tour).

Ciekawym rozwiązaniem byłoby tworzenie mapy w sposób losowy rozpoczynając nową rozgrywkę.