Проект Dungeons & Dragons: DrASCIIsang

Глава 1. Увод

1.1. Описание и идея на проекта

Проектът представлява RPG игра в тематиката на Dungeons & Dragons. Играчът се движи из генерирана карта, събира съкровища, бие се с чудовища и напредва през различни нива. Името DrASCIIsang е препратка към друга известна RPG игра (Drakensang Online), но в случая едно съществено нещо се отличава - графиката е изцяло конзолна.

1.2. Цел и задачи на разработката

Целта на разработката е да се създаде интерактивна и вълнуваща RPG игра, която включва:

- Генериране на карта лабиринт и разполагане на събития.
- Бойна система, включваща атаки, магии и използване на предмети.
- Събития като съкровища, чудовища и преминаване на нива.
- Реалистично и интуитивно управление на взаимодействията с играча.

Огромна тежест е поставена върху това в играта да бъде забавна за играене. Използвайки зададените от условието на проектна стойности и механики, играта е изключително трудна и неангажираща. За да се справя с този проблем съм използвал малко творческа свобода чрез добавянето на нови функционалности и нагаждане на правилни стойности за статистиките по време на бой.

В самата игра основна цел е събирането на жълтици, които в края на играта биват преброени. Отделните играчи след това се нареждат в класация според броя събрани жълтици. Събирането им изисква стратегическо мислене от страна на играча: той може да влиза по-често в бой да събира пари от драконите, рискувайки живота си в процеса, или да се опита да стигне до края на лабиринта, събирайки произволно генерираните съкровища по пътя. Целта на проекта е да направи всяка възможна стратегия еднакво приложима и конкурентноспособна. Това осигурява, че играта няма да омръзне веднага.

1.3. Структура на документацията

Документацията е структурирана в следните глави:

1. Увод
2. Преглед на предметната област
3. Проектиране
4. Реализация и тестване
5. Заключение

Глава 2. Преглед на предметната област

2.1. Основни дефиниции, концепции и алгоритми

Основни концепции включват ООП принципи за създаване на класове и методи, както и използването на стандартни библиотеки за работа с потоци и системни команди.

Най-често използваният алгоритъм в програмата е този за намиране на n-тото число от редицата на Фибоначи при зададена първа и втора стойност. Той служи за задаването на големината на картата, броя на чудовищата и броя на съкровищата.

Генерирането на пътища в картата на играта става посредством алгоритъм за търсене в дълбочина в правоъгълен масив от символи, който на всяка стъпка отива в произволна посока.

2.2. Дефиниране на проблеми и сложност на поставената задача

Основните проблеми включват генериране на динамична карта, създаване на бойна система и управление на различни типове събития. Сложността идва от необходимостта за синхронизация на различни компоненти и интеракцията им в реално време.

Най-големият проблем по време на разработката на проекта беше визуализацията на картата. Тъй като графиката е конзолна, много оптимизации трябваше да бъдат направени. Например, визуализацията на цялата карта наведнъж на всяка стъпка на играча е изключително бавен процес и дори се вижда как конзолата не смогва достатъчно бързо да трие и презаписва символите на всеки кадър.

Друг основен проблем беше визуализацията на картата при нейното уголемяване с постепенното напредване от ниво на ниво. Още преди 4 или 5 ниво картата става по-голяма от повечето екрани и не може да бъде визуализирана цялата наведнъж.

2.3. Подходи и методи за решаване на проблемите

Използваният подход включва обектно-ориентирано програмиране за модуларизация на функционалностите и лесно управление на промените. Всеки клас представлява отделна функционалност с ясно дефинирани методи и данни.

Проблема с графиката беше решен с написването на клас визуализатор (renderer), чието основно предназначение е да прави едно от две неща:

1. При разхождане из картата да се презаписват единствено символите около играча. Това става чрез промяна на координатите на четеца на самата конзола. Обаче това поражда друг проблем: ако картата стане по-голяма от екрана и съдържанието на конзолата трябва да се scroll-ва, то координатите на четеца се променят, защото те са отправни на горния ляв ъгъл на прозореца на конзолата и няма памет за това, което е останало нагоре след като се scroll-не. Затова имплементирах:
2. Възможност за разделяне на картата на парчета с предварително зададена големина и визуализиране на парче по парче. Това се оказа дефинитивно най-трудната част от проекта, защото е необходима и визуализация на малки части от съседните парчета. Това има много крайни случаи, като например когато картата не може да бъде разделена на равни парчета.

2.4. Потребителски и качествени изисквания

Потребителски изисквания като права, роли, статуси и т.н. няма. Единствено при начало на играта всеки може да избере какъв клас да бъде техният герой. Администраторски режим няма, защото в условието се иска само администратор да позволява да се генерират произволни нива при наличието на администраторски режим. Намерих това за излишно, защото по-важно за играта е всяка нова игра да бъде различна, без значение дали играчът е администратор или не. Още при натискане на NewGame(нова игра) се генерира нова карта, която никой досега не е обикалял.

Качествени изисквания включват:

- Интуитивен интерфейс.
- Стабилност и липса на грешки.
- Лесна разширяемост и поддръжка.

Броят на файловете и класовете са особено много, за да може всичко да е възможно най-модуларно и скалируемо.

Глава 3. Проектиране

3.1. Обща архитектура – ООП дизайн

Общата архитектура включва следните основни класове, групирани в следните групи:

Карта:
- Map: отговаря за генериране и управление на картата.
- Renderer: композиран вътре в Map. Отговаря за ефикасната визуализация на картата

Живи същества:
- Player и Monster: представляват игровите герои.
- Entity: клас родител на Player и Monster

Събития:
- CombatEvents и EventHandler: управление на бойни и други събития.

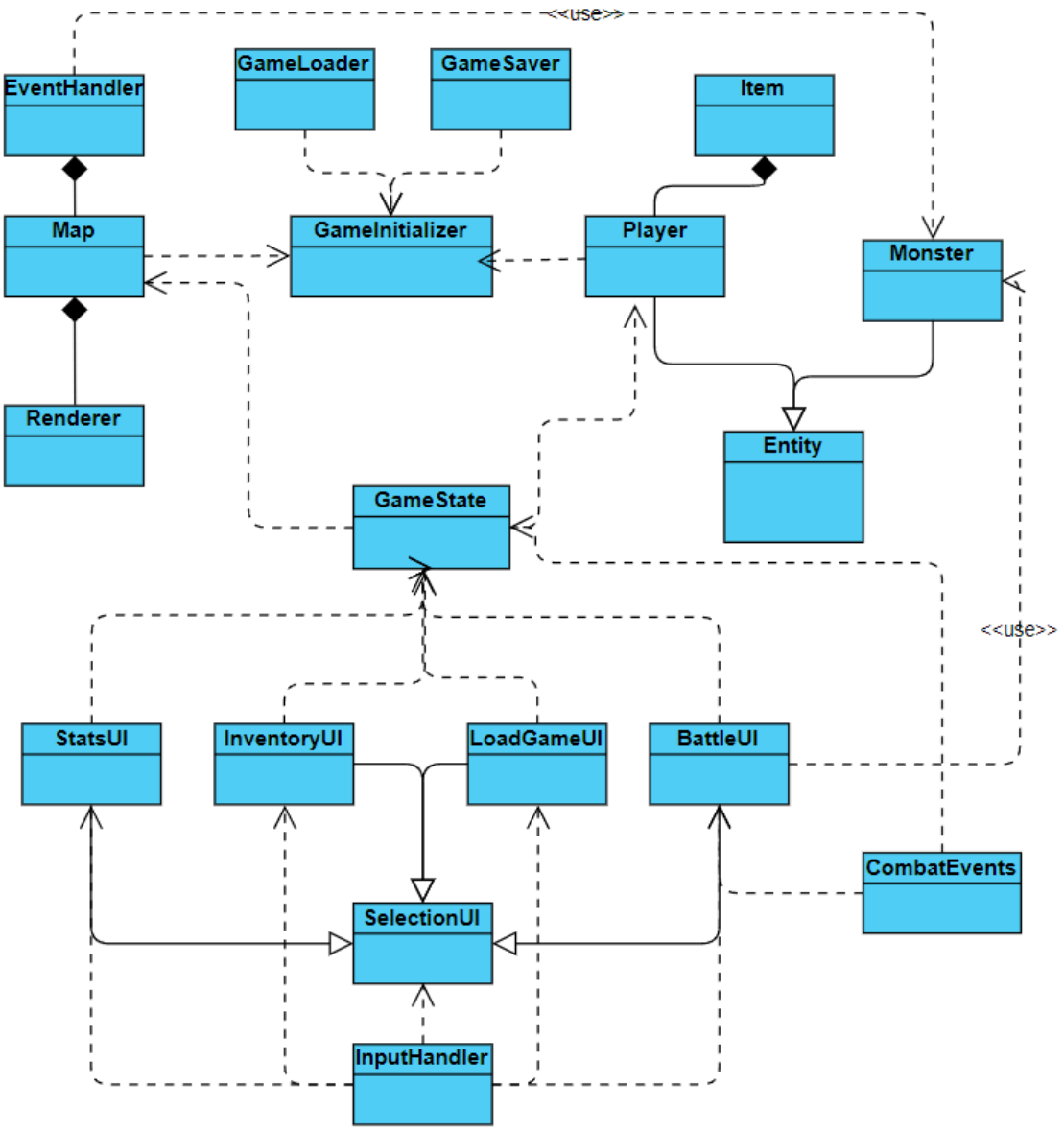
Потребителски интерфейс:
- BattleUI: интерфейс за бойната система.
- SelectionUI: списък от възможни стойности, една от която може да се избере
- InputHandler: обработва входовете от клавиатурата
- LoadGameUI: интерфейс за зареждане на прогрес
- StartMenuUI: интерфейс за начално меню; позволява започването на нова игра, зареждането на запазена игра и визуализация на класацията с най-богати играчи
- StatsUI: интерфейс за статистиките на героя
- InventoryUI: интерфейс за инвентара на героя

Предмети
- Item: представлява предмет, който може да бъде в инвентара на играча

Помощни:
- GameInitializer: отговаря за стартирането и приключването на играта
- Constants: съдържа важни константи
- GameSaver: логика за запазване на прогрес във файл
- GameLoader: логика за зареждане на прогес от файл
- GameState: съдържа ключови членове данни
- MathFunctions: съдържа помощни математически функции

3.2. Диаграми

Диаграмата на класовете показва взаимодействията между класовете:



Всеки клас съдържа много полета и методи, затова са показани само връзките между класовете: наследяване, композиция и зависимости.. Целта е да се онагледи общата архитектура на проекта.

По-подробни диаграми са включени в Doxygen документацията.

Глава 4. Реализация и тестване

4.1. Реализация на класове

Реализацията на класовете включва:

Map: генерира и поддържа текущото състояние на картата.
Player: държи информация за играча, като здраве, атаки и предмети.
Monster: подобен на класа Player, но с допълнителни функции за атака.
BattleUI: управлява визуализацията на боя.
CombatEvents: обработва различни бойни събития като атака и използване на предмети.
EventHandler: управлява събития като съкровища и преминаване на нива.

Подробности има в генерирания от Doxygen файл. Ето няколко важни моменти в реализацията на ключовите класове:

Пример 1 - създаване на лабиринт (Map)

```
void Map::CreatePath(const int i, const int j){ //Use DFS
    int directions[][2] = {{0,1}, {0,-1}, {-1,0}, {1,0}};
    std::vector<unsigned> visitOrder = {0,1,2,3};
    //out of boundary
    if(i < 0 || j < 0 || i >= height || j >= width) return ;
    //visited, go back to the coming direction, return
    if(matrix[i][j] == Constants::PATH) return ;

    //some neightbors are visited in addition to the coming direction,
return
    //this is to avoid circles in maze
    if(countVisitedNeighbor(i, j) > rand() % LABYRINTH +  1) return ;

    matrix[i][j] = Constants::PATH; // visited

    //shuffle the visitOrder
    MathFunctions::Shuffle(visitOrder);

    for (int k = 0; k < 4; ++k)
    {
        int ni = i + directions[visitOrder[k]][0];
        int nj = j + directions[visitOrder[k]][1];
        CreatePath(ni, nj);
    }
}
```

Пример 2 - Битка с чудовище (CombatEvents)

```
void CombatEvents::HandleSpellAttack(BattleUI& ui, bool& playerIsDead,
bool& monsterIsDead) {
    std::ostringstream msg;
    if (map->GetPlayer()->SpellAttack(*ui.getMonster())) {
        ui.Render();
        msg << map->GetPlayer()->GetName() << " has slain the dragon and
earned " << std::to_string(5 + map->GetFloor())
            << " gold and " << std::to_string(25 + map->GetFloor() * 3)
<< " xp";
        Alerts::Alert(msg);
        DropPotion(*map->GetPlayer());
        monsterIsDead = true;
    }
    else {
        ui.Render();
        msg << map->GetPlayer()->GetName() << " cast a " <<
map->GetPlayer()->GetSpell().GetName() << " on the dragon!";
        Alerts::BattleAlert(msg);
        Event event(map);
        event.MonsterAttack(map->GetPlayer(), ui.getMonster(),
playerIsDead);
        system("cls");
        ui.RenderGraphics();
        ui.Render();
    }
}
```

4.2. Управление на паметта и алгоритми

Оптимизираното управление на паметта включва използването на стандартни контейнерни класове и освобождаване на динамично заделена памет в класове като Map. Алгоритмите за визуализация и генерация на картата са реализирани с фокус върху ефективността и минимизиране на времето за изпълнение.

4.3. Планиране и тестване

Тестовите сценарии включват:

- Проверка на генерирането на картата.
- Симулиране на боеве и събития.
- Интеракции с играча и отговарящи действия на системата.
- Документални тестове с doctest гарантират коректното поведение на класовете и методите.

Примерен тест - удар на играч/чудовище

```
TEST_CASE("Entity::TakeHit") {
    Entity e;
    e.SetMaxHP(100);
    e.SetHP(100);
    e.SetArmor(20);

    SUBCASE("Entity survives hit") {
        CHECK(e.TakeHit(50) == false); // 50 - 20% of 50 = 40 damage
        CHECK(e.GetHP() == 60);
    }
    SUBCASE("Entity dies from hit") {
        CHECK(e.TakeHit(200) == true); // 200 - 20% of 200 = 160 damage
        CHECK(e.GetHP() == 0);
    }
}
```

Глава 5. Заключение

5.1. Обобщение на изпълнението на началните цели

Проектът изпълнява поставените цели за създаване на интерактивна RPG игра, наподобяваща D&D. С цел баланс и по-забавно преживяване по време на игра е добавена модифицирана инвентарна система и опцията за използване на отвари.

5.2. Насоки за бъдещо развитие и усъвършенстване

Бъдещите подобрения включват:

- Добавяне на повече видове чудовища и предмети.
- Добавянето на система за търговия на предмети.
- Симулиране на уникален стил на игра за всяка раса
- Разширяване на бойната система с нови умения и магии.
- Подобряване на интерфейса и визуализациите основно чрез добавяне на анимации по време на битка.
- Пресъздаването на играта, обаче с графична библиотека

Връзка към хранилището в Github:

Github хранилище