Втори проект по ООП

Тема 13 – Dungeons And Dragons

Стоян Димитров Малинин, 8-ма група, 0MI0800080

Връзка с GitHub - https://github.com/StoyanMalinin/DungeonsAndDragons

**Използване на документацията**

Идеята на тази част от документацията е да опише замисъла зад имплементацията. Също и да се покажат примери и да се направят коментари. Целта на документацията, генерирана от **doxygen** е просто да има лесен начин да се навигира по кода и да се гледа неговата структура.

**Описание на задачата**

Целта на проекта беше да се създаде удобна и стабилна за ползване реализация на играта Dungeons And Dragons. Трябваше да се имплементира конзолен интерфейс, който да поддържа набор от команди, както и самата игра също конзолно. Основни части от реализирането на играта бяха – изчисляване на стойности за отделни нива, генериране на карти за ниво възможност за различно поведение на отделните компоненти, изграждане на система за битки и система за обмен на предмети.

**Подход за решение**

Приложението е написано много обектно ориентирано, което е добър избор за този вид задача, понеже има много елементи с подобно поведение, което би било добре да може да се преизползва.

**Архитектура на приложението**

**Entry Point**

Цялата дейност на програмата се контролира от класът **Application.** Той следи за това кои команди се пускат и също така обработва грешките, които са се случили по време на тяхното изпълнение. За да стане комуникацията с потребителя, **Application** много често се обръща към **UIHandler**, от който изисква някакъв изход или вход на данни. Някои от другите обекти на по-ниско ниво в имплементацията също имат връзка с **UIHandler.**

**Игрални обекти**

В играта има няколко основни вида обекти, повечето от които са наследници на класът **GameEntity**. За по-добра яснота е добре да се прави справка с UML диаграмата на проекта. Основната характеристика на **GameEntity** обектите е, че могат да си взаимодействат(**interact-**ват).

* **GameItem –** обект, който описва игрален предмет. Това е йерархията, която не е наследник на **GameEntity,** понеже предметите не **interact-**ват.
  + **Armor –** броня
  + **Spell –** заклинание
  + **Weapon –** оръжие
* **TileEntity –** обект, който е пригоден да стои на игрална карта. Съдържа само основна информация за местоположението си, начина си на визуализация и дали може да се „влезе в него“
  + **EmptyTile**
  + **ExitTile –** изход от нивото
  + **WallTile**
  + **Treasure –** клас „съкровище“. Съдържа точно един предмет, който може да бъде взет от играча(това е вида интеракция на класа). Също така съдържа информация дали предмета вече е взет.
    - **ArmorTreasure –** съдържа броня(**Armor**)
    - **SpellTreasure –** съдържа заклинание(**Spell**)
    - **WeaponTreasure –** съдържа оръжние(**Weapon)**
  + **MovableTileEntity –** игрален елемент на картата, който може да се движи
* **FightableEntity –** игрален обект, който има характеристики и възможност да участва в битки. Това е и поведението, което се очаква от него при интеракция.
* **Player –** класът за играч. Наследява **FightableEntity** и **MovableTileEntity.** Пази нужната информация за играча, описана от условието
* **Dragon –** обект за дракон. Наследява **FightableEntity** и **TileEntity.**  Единственото, което може да прави е да се движи.

**Решения за действия от игралните обекти - контролери**

Ако някой обект може да прави нещо, за което се изисква някакво решение, то най-вероятно този обект съдържа някакъв вид **контролер**. Идеята на контролерите е да казват на обекта какво решение да вземе на база на това те какъв вид са и каква информация имат.

* **FightController** – решава избора на това каква атака да използва едно **FightableEntity** – силова или магическа
  + **UIFightController** – взема решения на база на потребителския избор
  + **RandomFightController** – взема решения на случаен принцип
* **MoveController** – взема решения за това как едно **MovableTileEntity** да се движи
  + **OnlyDownMoveController** – движи се само надолу
  + **UIMoveController** – взема решения на база на потребителския избор
* **PointsDistributionController** – взема решение как играча да си разпредели точките след края на нивото
  + **EvenPointsDistributionController** – разпределя точките поравно
  + **UIPointsDistributionController** – разпределя точките на база на потребителския избор

Тази система е много гъвкава и позволява с много малко усилия да се правят неща от рода на: създаване на дракон, който е управляван от човек. Друг интересен вариант би бил да се направи играч, който се управлява автоматично.

**Взаимодействия между игралните обекти – InteractionMaster**

Всички обекти от вида **GameEntity** могат да си взаимодействат помежду си. Тъй като йерархията е много голяма, а интеракцията между два обекта в някакъв смисъл двуизмерна задача, то разчитането само на полиморфизъм няма да е достатъчно. По тази причина са направени тези „помощни“ обекти, които при всяка интеракция между два игрални обекта да следят как те могат да си взаимодействат

* **InteractionMaster –** абстрактен клас, който следи за интеракциите между обектите
  + **FightMaster –** следи за битките между FightableEntities
  + **ItemExchangeMaster –** следи за това предметите от съкровищата да се предлагат на играча

Фактът, че различни обекти могат да се „свързват“ към различни **InteractionMaster**-и дава възможност да се забрани взаимоотношението между някои обекти като се постигне ефект на разделяне на слоеве.

**Създаване на ниво**

Информацията за едно ниво се съдържа в обектите от тип **Level.** Там се съдържа информация за номера на нивото, както и за картата. В класа също се съдържа логиката, по-която се развиват стойностите в играта.

* Размери на карта, брой дракони, брой съкровища – растат със скоростта на линейно-рекурентна редица като тази на Фибоначи
* Сила, здраве и магия на дракони, коефициенти на предметите от съкровища – растат със скоростта на леко модифицирана версия на Sigmoid function.

**Създаване на карта**

Информацията за карта на ниво се съдържа в обектите от тип GameMap. Най-трудната част от създаването на една карта е да се определи кои ще са празните позиции и кои ще са тези, в които има стена. Грубо казано, картата се състои от стаи(правоъгълници от празни клетки, успоредни на картата) и коридори(произволни пътища свързващи стаите). Цялото нещо се ръководи от алгоритъм, който прави **Space Partitioning,** разделяйки картата на сектори и свързвайки ги след това.

**Recovery**

По време на игра между всяко придвижване на играча, програмата запазва моментното състояние на играта във файл, за да може при инцидент, играта да бъде възстановена.

|  |  |
| --- | --- |
| Стъпка 0 | Стъпка 1 |
| Стъпка 2 | Стъпка 3 |
| Стъпка 4 | Стъпка 5 |
| Стъпка 6 |  |

**Използване на приложението**

**Силно препоръчително е при компилация да си настроите размера на стека да бъде поне 4MB, за да нямате проблеми с генерирането на нива. Малки нива би трябвало да може да се генерират и без това.**

Информация се въвежда по два основни начина – чрез командни параметри и чрез пряко подканване чрез диалогово съобщение. В повечето команди се използват и двете. Тук има описание на основните команди. **Входът винаги е case-sensitive.**

|  |  |
| --- | --- |
| Синтаксис на команда | Описание на функционалност |
| ls [относителен път към директория = „.“] | Печата всички файлове и папки в посочената директория |
| exit | Напускане на приложението |
| generatePlayer [файл] | Генерира играч по желани данни от потребителя и запазва информацията във [файл] |
| generateLevel [файл] | Генерира ниво по желани данни от потребителя и запазва информацията във [файл] |
| dnd [файл1] [файл2] | Стартира игра на D&D с играча от [файл1] и нивото от [файл2] |
| recover [файл1] [файл2] [файл3] | Възстановява информацията от recovery файла [файл1], като записва информацията за играча във [файл2] и тази за нивото във [файл3]. |
| changeBackupFile [файл] | Прави така, че вече recovery информацията ще се пази във [файл]. Файлът по подразбиране е backup.txt. |

По време на самата игра трябва да се спазват инструкциите, които биват показани. В общия случай трябва да се задава посоката на движение на играча. По време на битка трябва да се правят избори за вида атака, а при интеракция със съкровище трябва да се вземе решение дали играчът да вземе новия item.

**Тестване**

Приложението е написано по такъв начин, че много голяма част от него може да се тества автоматично чрез unit tests. Също така за визуализация и дебъгване във файла Sandbox.h има някои предварително подготвени сценарии.

**Подобрения**

Може да се направи по-обстойно тестване на играта, което да включва и следене на разход на памет и време, за да се оптимизира. Работата с файлове би могла да се направи по-абстрактна. Може да се усложни играта като се добавят повече чудовища, предмети и по-разнообразно създаване на нива.

**Демо**

При стартиране на приложението без да има други файлове може да се направят следните действия с цел демонстрация.

**Създаване на играч**

Text

Description automatically generated

**Създаване на ниво**

Text

Description automatically generated

**Стартиране на игра с тези параметри**

Graphical user interface, text

Description automatically generated

**Преместване до първо съкровище и вземане на item-a**

Един възможен начин е да се даде два пъти надясно.

Text

Description automatically generated

**Изместване извън съкровището и изключване на програмата**

Text

Description automatically generated with medium confidence

**Стартиране на програмата и възстановяване на състоянието ѝ**



**Отново стартиране на играта**

Text

Description automatically generated

**Започване на бой с дракона долу вдясно**

Няма по-подробно описание, понеже ходът на битката се определя на случаен принцип.

**Придвижване до изхода (ако героя е оцелял битката)**

Възможен начин е да се даде 4 пъти надолу.