ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
към СОФИЙСКИ УНИВЕРСИТЕТ “СВ. КЛИМЕНТ ОХРИДСКИ”

КУРСОВА РАБОТА

Тема: The Game of Life

Студент: *Мартин Дацев*

Факултетен номер: *45666*

СОФИЯ

2020

Глава 1. Увод

1.1. Описание и идея на проекта

Играта *Game of Life* е клетъчен автомат измислен от Джон Конуей през 1970 година [1]. Идеята на проекта се имплементира вариация на тази игра.

Да има "дъска" състояща се от M x N квадратни клетки. Във всяка клетка или има живо същество или не. Две клетки наричаме съседни, ако имат обща стена или ъгъл. Живите същества могат да бъдат от различни племена (т.е. условно можем да ги наречем Варвари, Рицари, Граждани и т.н). Всяко племе се визуализира с различен цвят върху дъската. Дъската има начално състояние. То определя в кои клетки има живот, в кои не и кое същество от кое племе е. Играта се състои от безброй много ходове, като на всеки ход дъската се променя по дадени правила.

1.2. Цел и задачи на разработката

Целта на проекта е да може да се разглежда и изучава играта като това изисква визуализация на дъската, възможност за манипулация на дъската, и правилно ъпдейтване на самата дъска. Правилата по които се променя дъската са следните:

- Всяка жива клетка с по-малко от две живи (независимо от племето) съседни клетки умира (от самота).
- Всяка жива клетка с повече от три живи (независимо от племето) съседни клетки умира (от пренаселеност).
- Всяка жива клетка с две или три живи (от същото племе) съседни клетки остава жива и на следващата итерация.
- Съдбата на всяка жива клетка, която не удовлетворява никое горните условия, се решава от случайността (шанс 50% за живот и 50% за смърт).
- Всяка мъртва клетка с точно три живи (от същото племе) съседни клетки се превръща в жива клетка (от съответното племе).

След края на всеки ход дъската трябва да се визуализира на екрана. Между два последователни хода трябва да се изчака известно време за да може човек да наблюдава развитието на играта.

Трябва да има възможност за паузиране на играта и манипулация на полето като се махат/добавят същества и да може да се правят стъпки по един ход.

1.3. Структура на документацията

Документацията е разделена на глави които описват различните части от разработката:

1. Увод - описание и идея на проекта, цели и задачи, структура на документацията

2. Преглед на предметната област - дефиниции, концепции, алгоритми, проблеми, методи,
3. Проектиране - обща архитектура, диаграми на поведение
4. Реализация, тестване - реализация на класове, алгоритми, оптимизации, тестови сценарии
5. Заключение - обобщение на изпълненото и бъдещо развитие
6. Използвана литература

Глава 2. Преглед на предметната област

2.1. Основни дефиниции, концепции и алгоритми, които ще бъдат използвани

Дъската на играта представлява правилна решетка от M x N квадратни клетки, всяка от която е в едно от 2 състояния - жива или мъртва. Всяка от живите клетки принадлежи към дадено “племе” от краен брой племена. Всяка клетка има 8 на брой съседа - тези с които споделя стена или ъгъл. Задава се начално състояние на дъската и от там нататък времето се дели на дискретни стъпки (t=1, 2…). За всяка следваща стъпка (t се увеличава с 1) се създава ново състояние на дъската по определените правила. Важно е правилата да се приложат едновременно за цялата дъска, за да може новото състояние да зависи само от старото, а не от междувременни изчисления.

2.2. Дефиниране на проблеми и сложност на поставената задача

Основните проблеми/задачи които трябва да се изпълнят са визуализацията на дъската, възможност за манипулация на дъската, и правилно ъпдейтване на самата дъска.

2.3. Методи за решаване на поставените проблемите

За визуализацията на дъската се използва библиотеката за компютърна графика на C++ - SFML. За бързо рендиране на голям брой клетки се използват *Vertex Arrays*, и клетките се рендират в един *Draw Call.* За манипулация на дъската се използва функционалността на същата библиотека за вземане на инпут от клавиатура и мишка. За имплементацията и ъпдейтването на самата дъска са използвани набор от класове с методи за играта, клетки, племена.

2.4 Потребителски изисквания и качествени изисквания

За потребителите е важно визуализацията да е разбираема, да има възможност за ъпдейтване на дъската стъпка по стъпка за по-лесно изучаване, паузиране на играта и за манипулация на

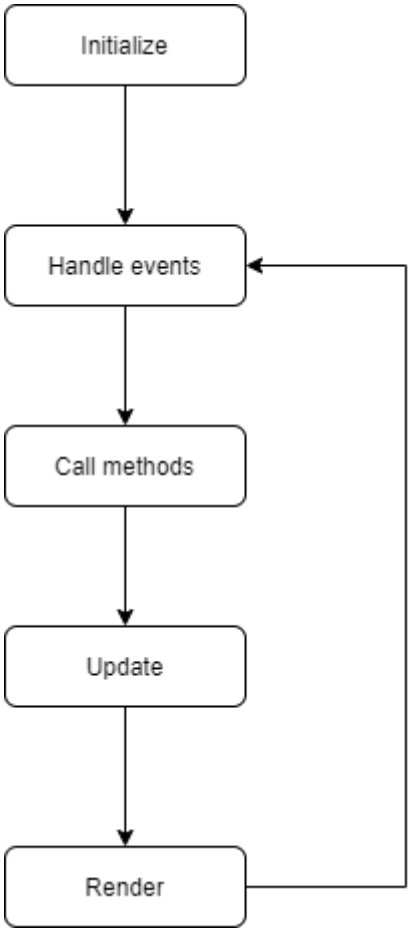
дъската. Поради високо скалируемата природа на играта трябва да могат да се поддържат голям брой клетки без голямо намаление на бързодействието (поне 100 000 клетки).

Глава 3. Проектиране

3.1. Обща архитектура

Основната функционалност на играта като рендиране и ъпдейтване на дъската е имплементирана в класа GameOfLife. В main функцията има една инстанция на този клас и един *SFML Event Loop,* който хендълва интеракции с мишката и клавиатурата и извиква съответните методи на инстанцията на GameOfLife. Реализирани са помощни класове съответно за клетките и племената.

3.2. Диаграми на поведение



Фигура 3.2. Диаграма на поведение

В началото на изпълнението се инициализират ресурсите като прозорец и инстанция на играта, като след това започват да се хендълват евенти и да се извикват съответните методи за тях и се ъпдейтва дъската. Рендира се дъската върху прозореца и се започва отначало.

Глава 4. Реализация, тестване

4.1. Реализация на класове

```
class GameOfLife {
    GameOfLife(int height, int width, int n_tribes, int cell_size);
    void update();
    void render(sf::RenderWindow& window);
    void click(int x, int y);
    void switch_click_tribe();
    void switch_click_kill();
};
```

Класа `GameOfLife` имплементира конструктор, с който се задава височината и ширината на дъската, броя племена и големината на една клетка. Метода `update` преминава състоянието на дъската към следващата итерация. Метода `render` рендира дъската върху подадения прозорец. Метода `click` изпълнява клик върху клетка на дадените координати като това какво прави кликът се контролира от `switch_click_kill` дали убива клетката и `switch_click_tribe` кое племе да съживи в клетката.

4.2. Управление на паметта и алгоритми. Оптимизации.

```
void GameOfLife::update() {
    std::vector<Cell> old_cells = cells;
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            Cell cell = get_cell(old_cells, x, y);
            std::array<Cell, 8> neighs = get_neighbours(old_cells, x, y);
            get_cell(cells, x, y) = get_new_cell_state(cell, neighs);
        }
    }
}
```

Във метода `update` е имплементиран ъпдейтът на дъската, като първо се копира цялата дъска, и от копието се вземат съседите за да може новото състояние да зависи само от старото, а не от междувременни изчисления. Правилата за всяка клетка са имплементирани в метода `Cell GameOfLife::get_new_cell_state(Cell cell, const std::array<Cell, 8>& neighs);`

Една важна оптимизация е рисуването на клетките да не става 1 по 1, защото това отнема прекалено много време при голямо количество клетки. Това е постигнато в метода за рендиране чрез `sf::VertexArray(sf::Quads, cells.size() * 4)` който се попълва с координатите на клетките и се рисува с един *draw call.*

4.3. Планиране, описание и създаване на тестови сценарии

Тествани са всички главни функционалности на проекта, като за тестване ъпдейтването по правилата е използвана функционалността за пауза, манипулация на клетките и ъпдейтване стъпка по стъпка, за да се тестват различни конфигурации от племена. Също така има сравнение с други имплементации на Game of Life и сравнение на поведението на различни конфигурации като например *glider*.

Глава 5. Заключение

5.1. Обобщение на изпълнението на началните цели

Изпълнени са всички цели описани в текущата документация. Може да се разглежда и изучава играта като чрез визуализация на дъската, възможност за манипулация на дъската, и ъпдейтване на дъската по определените правила.

5.2. Насоки за бъдещо развитие и усъвършенстване

Възможни насоки за развитие са по-добър интерфейс за работа с играта и възможност за различни правила за клетките.

Използвана литература

[1] MATHEMATICAL GAMES The fantastic combinations of John Conway's new solitaire game "life" https://web.stanford.edu/class/sts145/Library/life.pdf

[2] SFML Documentation https://www.sfml-dev.org/documentation/2.5.1/index.php