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

КУРСОВА РАБОТА

Тема: Игра 2048

Студент: *Мартин Дацев*

Факултетен номер: *45666*

СОФИЯ

2020

Глава 1. Увод

1.1. Описание и идея на проекта

Идеята на проекта е да се реализира игра “2048”. Играта представлява дъска с плочки, които имат числови стойности във вида 2^n, където n е цяло число, по-голямо от 0. Идеята на играта е плочките да могат да се плъзгат и да се комбинират плочки с еднаква стойност в една плочка с 2 пъти по голяма стойност.

1.2. Цел и задачи на разработката

Целта на проекта е да може да се играе играта като са изпълнени следните функционалности:

- Визуализация на дъската и плочките върху нея.
- Възможност за изпълняване на 4 възможни хода - плъзгане на всички плочки в една от четирите посоки
- Комбиниране на еднакви плочки в 2 пъти по голяма
- Появяване на нова плочка на всеки успешен ход.
- Броене на резултата (сумата на комбинираните плочки)
- Броене на най-висок резултат, който да се запазва между игри
- Undo - връщане на предишния ход
- Redo - обратното на undo
- Анимация на местенето на плочките
- Играта да свършва когато няма възможни ходове

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 представлява квадратна решетка от клетки като някои от тях могат да съдържат плочки, които имат числови стойности във вида 2^n, където n е цяло число, по-голямо от 0. Всеки ход представлява движение на всички плочки в една посока, като плочката се движи

докато стигне стената или друга плочка. Ако стигне плочка със същата стойност се слива с нея в плочка с 2 пъти по-голяма стойност. Не може на един ход дадена плочка да се слее повече от веднъж.

2.2. Дефиниране на проблеми и сложност на поставената задача

Основните проблеми/задачи които трябва да се изпълнят са визуализация на дъската и плочките върху нея, алгоритъма за изпълнение на ходовете, анимацията за местене на плочките, undo и redo функционалностите.

2.3. Методи за решаване на поставените проблемите

За визуализацията на дъската се използва библиотеката за компютърна графика на C++ - SFML. За манипулация на дъската се използва функционалността на същата библиотека за вземане на инпут от клавиатурата. За имплементацията и ъпдейтването на самата дъска са използвани набор от класове с методи за играта, плочките и други помощни класове.

2.4 Потребителски изисквания и качествени изисквания

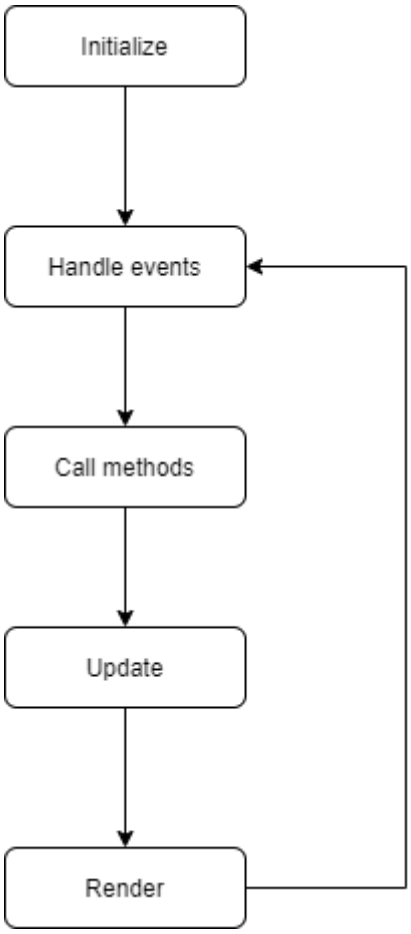
За потребителите е важно визуализацията да е разбираема, да има анимации, играта да работи както се очаква от изискванията, да има undo/redo и броене на резултата.

Глава 3. Проектиране

3.1. Обща архитектура

Основната функционалност на играта като рендиране и ъпдейтване на дъската е имплементирана в класа Game. В main функцията има една инстанция на този клас и един *SFML Event Loop,* който хендълва входа от клавиатурата и извиква съответните методи на инстанцията на Game.

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 Game {
    Game(int height, int width, int scale, float padding, int startTiles);
    void render(sf::RenderWindow& window);
    bool move(int xDir, int yDir, bool dryRun = false);
    void undo();
    void redo();
    void restart();
    void close();
};
```

Класа `Game` имплементира конструктор, с който се задава височината и ширината на дъската, колко големи да са плочките и разстоянието между тях и броя начални плочки. и големината на една клетка. Метода `move` изпълнява ход в зададената посока. Метода `render` рендира дъската върху подадения прозорец. Методите `undo, redo` изпълняват съответните действия. `restart` рестартира играта. `Close` освобождава ресурсите и записва данните на диска (най-високия резултат).

4.2. Управление на паметта и алгоритми. Оптимизации.

```
bool Game::move(int xDir, int yDir, bool dryRun) {
    UndoState original = getUndoState();
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            tiles[x][y].reset();
        }
    }
    int farthest = tiles.size() - 1;
    bool moved = false;
    for (int x = (xDir < 0) ? 0 : farthest; x >= 0 && x <= farthest; x -=
(xDir ? xDir : 1)) {
        for (int y = (yDir < 0) ? 0 : farthest; y >= 0 && y <= farthest; y -=
(yDir ? yDir : 1)) {
            Tile& tile = tiles[x][y];
            int value = tile.get_value();
            if (value == 0) {
                continue;
            }
            Tile* lastEmpty = nullptr;
            for (int xNext = x + xDir, yNext = y + yDir; xNext >= 0 && xNext
<= farthest && yNext >= 0 && yNext <= farthest; xNext += xDir, yNext += yDir)
{
                Tile& nextTile = tiles[xNext][yNext];
                if (nextTile.get_value() == value) {
                    moved = true;
                    nextTile.mergeWith(tile);
                    addScore(value * 2);
                    lastEmpty = nullptr;
                    break;
                } else if (nextTile.get_value() == 0) {
                    lastEmpty = &nextTile;
                } else {
                    break;
                }
            }
            if (lastEmpty) {
                moved = true;
                lastEmpty->mergeWith(tile);
```

```
                }
            }
        }
        if (dryRun) {
            setState(original);
            return moved;
        }
        if (moved) {
            spawnTile();
            animClock.restart();
            pushUndoState();
            animLoc = undoLoc - 1;
        }
        return moved;
    }
```

Основната логика е в метода move. Важната част е че клетките започват да се итерират от най-далечната в посока на хода, за да може да се сливат правилно. След това за всяка клетка се намира най-далечното място където може да отиде и се слива там.

4.3. Планиране, описание и създаване на тестови сценарии

Тествани са всички главни функционалности на проекта. Също така има сравнение с други имплементации на 2048 и сравнение на поведението на различни конфигурации от плочки и цялостното поведение на играта.

Глава 5. Заключение

5.1. Обобщение на изпълнението на началните цели

Изпълнени са всички цели описани в текущата документация. Има визуализация на дъската и плочките върху нея, възможност за изпълняване на ходовете, и поведението на играта, броене на резултата и най-високия резултат, undo/redo и анимация на всички видове ходове и undo/redo.

5.2. Насоки за бъдещо развитие и усъвършенстване

За бъдещо развитие може да се направят повече game modes и запазване на повече статистики за резултата и играта.

Използвана литература

[1] Игра 2048 https://play2048.co/

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