ИГРА НА ПАМЕТ

ФИНАЛЕН ПРОЕКТ (EGT)

ИГРА НА ПАМЕТ
ПРАВИЛА НА ИГРАТА

- Играчът отваря последователно две карти.
- При обръщане на две еднакви карти, те остават отворени и играчът прави следващ избор от други две карти.
- Ако обърнатите карти на са еднакви, те се затварят и играчът прави следващ ход от обръщане на други две карти.
- Играта приключва, когато се отворят всички карти.

ИГРА НА ПАМЕТ: НАСТРОЙКИ

- Име на играча – трябва да може да се въвежда с клавиатурата.
- Фиксирано време за нареждане (1, 2:30, 5 и т.н. минути) или неограничено време.
- Динамичен размер на игровата зона – 4x4, 6x6, 8x8 - избира се на случаен принцип.
- Статистика – рекорди за всеки размер на игровата зона – име на играча, време за нареждане, брой грешки, брой ходове.

ИГРА НА ПАМЕТ
ЗВУЦИ

- При грешно отворен чифт;
- При правилно отворен чифт;
- При завършване на играта;
- При завършване на играта с нов рекорд;
- При загуба на играта;

ИГРА НА ПАМЕТ

ДОПЪЛНИТЕЛНИ ИЗИСКВАНИЯ

- Бутон, който да скрива / показва вече отворените карти.
- Бутон, който да скрива / показва информативния панел за изминалото време, направени ходове и грешки.
- Възстановяване на състоянието на текущата игра, ако излезем от нея (ходове, грешки, изминато време, избрани опции от играча)
- Записване и четене на рекордите от файл.

ПЛАН ЗА ИЗПЪЛНЕНИЕ
СТЪПКИ - ЧАСТ 1

- Създаваме проект **MemoryGame_EGT_FinalProject** и настройваме **SDL** библиотеките - SDL_image, SDL-ttf, SDL_mixer
- Създаваме клас **main**, където ще извикваме методите (от class Game), които ще се преповтарят, докато играта продължава. Тук настройваме размерите на екрана и задаваме последователността на изпълнението на методите.

ПЛАН ЗА ИЗПЪЛНЕНИЕ
СТЪПКИ - ЧАСТ 2
(**CLASS GAME**)

Създаваме клас **Game**, който има за цел да инициализира прозореца, да менажира ивентите, да ъпдейдва и визуализира дъската (class Board), докато играта продължава и да почиства паметта, след като играта приключи.

CLASS GAME (1):

- В метода void **Game::handleEvents()** менажираме ивентите чрез switch case:

case SDL_QUIT: Ако се кликне върху X играта ще приключи;

case SDL_MOUSEBUTTONDOWN: извиква функцията

board->mouseClicking(), чиято цел е да, да визуализира съответното изображение при кликване върху определен квадрант .

CLASS GAME (2):

- В метода **void Game::update()** се ъпдейтва конкретната текстура на борда, където се оперира с картите.
- Чрез метода **void Game::render(...)** се визуализират картите от борда.
- Извикваме метода void **Game::clean()**, след като играта е приключила, за да почисти паметта (дисторйваме прозореца и визуализацията)

ПЛАН ЗА ИЗПЪЛНЕНИЕ
СТЪПКИ - ЧАСТ 3
(**CLASS TAXTUREMANAGER**)

Създаваме клас **TaxtureManager,** в който задаваме метода **static TextureManager* s_tInstance,** за да не е нужно да създаваме обект в класа GameObject, за да достигнем до него.

- В метода **bool TextureManager::loadTexture(…)** , зареждаме текстурата, която ще можем да използваме в класовете Card и Board.

ПЛАН ЗА ИЗПЪЛНЕНИЕ
СТЪПКИ - ЧАСТ 4
(**CLASS CARD**)

Създаваме клас **Card**, който се грижи за конкретната карта.

- Създаваме **конструктор** с параметри – текстура на лицето и гърба на картата, име на текстурата, координати x и y и булева променлива (isFace), чрез която ще можем да обръщаме картата с лице и гръб.
- Създаваме сетери и гетери за булевата променлива isFace и за името на картата.

CLASS CARD (2)

- В методите **void Card::updateBack()** и

void Card::updateFace() задаваме параметрите на текстурата на лицето и гърба на картата) и визуализираме обектите.

- Чрез метода **void Card::renderCard()** визуализираме създаденият обект (ако картата е с лице се визуализира една текстура, ако е с гръб - друга)
- Създаваме метод **void Card::setPos(int x, int y),** който ще използваме при рандомизирането на картите.

ПЛАН ЗА ИЗПЪЛНЕНИЕ
СТЪПКИ - ЧАСТ 5
(**CLASS PLAYER**)

- Създаваме клас **Player**, в който ще се съхранява информацията, съпътстваща играча – име, направено ходове, направени грешки, точки, време за завършване на играта
- Създаваме **гетери** за private променливите (std::string playerName, double timeToSolve, int mistakes, int moves, int points;
- Създаваме методи които да добавят по една точка съответно за всеки ход, точка и грешка.

ПЛАН ЗА ИЗПЪЛНЕНИЕ
СТЪПКИ - ЧАСТ 6
(**CLASS SOUNDMANAGER**)

Създаваме singleton class **SoundManager**, който менажира звуците, използвани по време на играта. Този клас може да има само една инстанция

- Функцията static SoundManager* Instance() позволява създаването на инстанцията и запазва конструктора частен. Това гарантира, тя ще е винаги е достъпна от всяка част на програмата.

CLASS SOUNDMANAGER (2)

- Създаваме метод за зареждане на звуковия файл (**bool SoundManager::load(…)**), с параметри името, id и типа на файла. Под тип се има предвид дали той ще е музикален или звуков, за да може програмата да го разпознава.
- Създаваме методи за пускане на музика и звук void SoundManager::playSound(…) и void SoundManager::playMusic(…).

ПЛАН ЗА ИЗПЪЛНЕНИЕ
СТЪПКИ - ЧАСТ 7
(**CLASS BOARD**)

В класа **Board** ще оперираме със самите карти, играч и звук.

- Създаваме **конструктор** с параметър текстурата на борда.

В него:

- зареждаме текстурата на борда;
- инициализираме всяка карта, която ще използваме (за борд с полета 4x4 ще са нужни 8 карти, всяка от които да се повтаря два пъти. Зареждаме ги с гръбчета на горе по подразбиране.

CLASS BOARD (2) - КОНСТРУКТОР

- Добавяме картите в двумерен масив, като задаваме коя карта на кое място ще се намира.
- Създаваме играч, на който ще се изчислява статистиката по време на играта.
- Създаваме нова карта (current Card), чрез която ще сравняваме дали две отворени карти са еднакви. Сетваме дифолтни стойности на името и булевата променлива, която отговаря за обръщането на картата.

CLASS BOARD (3) - КОНСТРУКТОР

- **srand(time(0))** ще рандомизира подредбата на картите върху борда при всяко ново зареждане на играта.
- С for-цикъл поставяме картите на рандом места в масива и отново с for-цикъл сетваме позицията върху борда на всяка от тях, като позицията е съобразена с размера на самите карти : deckOfCards[i][j].setPos(j * 200, i * 200);
- Зареждаме звуците, които ще се използват по време на играта.

CLASS BOARD (4)

- Методът void **Board::renderCard()** съдържа спомагателната логика.
- В метода void **Board::renderCard()** визуализираме борда, както и масивът от карти, поставени на рандом позиции. Това от своя страна се случва чрез извикване на метода **void Board::cardsArrRender()**, където с for- цикъл визуализираме всяка карта отделно.

CLASS BOARD (5) - MOUSE CLICKING

В метода **void Board::mouseClicking()** се съдържа основната логика на играта.

- В началото на метода се задава типа на ивента, а именно кликване с мишката.
- Задаваме размера на екрана и x и y позицията на кликването.
- променливите indexX и indexY използваме за да отворим конкретната картинка, намираща се в тези координати:

  mx / 200 = xpos (0, 1, 2, 3) и my / 200 = ypos (0, 1, 2, 3)

  deckOfCards[indexY][indexX]

CLASS BOARD (6) - MOUSE CLICKING

- Използваме **if-else** конструкция, за да проверим дали името на картите съвпада.
- За целта тази конструкция ще бъде вмъкната в else на друга конструкция, с която проверяваме, дали временната карта има присвоена стойности за име. Ако не е – в if-statement временната карта взима тази стойност от картата, която стои на позицията, кликната с мишката.

CLASS BOARD (7) - MOUSE CLICKING

- При второто кликване с мишката добавяме 1 към ходовете на играча и отваряме нова карта и сравняваме името и с това на временната карта.
- Ако двете имена съвпадат, се изписва подходящо съобщение и се чува съответния звук за победа и двете карти остават отворени. Играчът получава точка.
- Ако двете имена не съвпадат, се изписва подходящо съобщение и се чува съответния звук за загуба и картите следва да се обърнат с гърбовете си (TODO). На играчът се добавя 1 нова грешка.

CLASS BOARD (8) - MOUSE CLICKING

- TODO: Играта приключва, когато картите на всички позиции са обърнати с лице.

TODO

- **Настройки на играта:**
- Име на играча – трябва да може да се въвежда с клавиатурата.
- Фиксирано време за нареждане (1, 2:30, 5 и т.н. минути) или неограничено време.
- Динамичен размер на игровата зона – 4x4, 6x6, 8x8 - избира се на случаен принцип.
- Статистика – рекорди за всеки размер на игровата зона – име на играча, време за нареждане, брой грешки, брой ходове.

TODO

- Бутон, който да скрива / показва вече отворените карти.
- Бутон, който да скрива / показва информативния панел за изминалото време, направени ходове и грешки.
- Възстановяване на състоянието на текущата игра, ако излезем от нея (ходове, грешки, изминато време, избрани опции от играча)
- Записване и четене на рекордите от файл.