Сборка

Сборка производилась на Win10 x64 с использованием C++20 & CMake & Ninja & MinGW 8.1 x64 в среде QtCreator.
Также проверялась сборка на VS 2022 (сгенерированный из CMakeLists.txt) & MSVC 143.
CMakeLists.txt ориентирован на Windows, но под Linux, в теории, тоже должно работать, если все библиотеки SDL установлены, проверить возможности нет.

Запуск

В директории /build находится исполняемый файл /SDL2Test.exe с библиотеками SDL2.dll & SDL2_ttf.dll.
Также необходимо убедиться в наличии директории /images с файлами текстур и файла шрифта JetBrainsMono-Regular.ttf в директории с исполняемым файлом.
Запуск проверялся также на Win10 x64.

Описание исходного кода

Далее будет приведено краткое описание основных компонентов программы.

MainWindow	
bool **init**();	Инициализирует все компоненты SDL и игровой логики
bool **loop**();	Запускает бесконечный цикл рендера и обработки событий
void **startDrums**();	Запуск прокрутки барабанов
void **render**(Stuff::FrameInfo **frameInfo**);	Рендер всех игровых компонентов

Button	
Button(SDL_Renderer ***renderer**, *const* std::string &**file**, *const* SDL_Rect &**pos**, std::function< void(void) > &**callback**);	Конструктор, принимающий на вход - рендер - файл для текстуры - положение кнопки - функцию обратного вызова
void render(float timeStep);	Отрисовка кнопки на экране. При отрисовке меняется прозрачность текстуры от minTransparency до maxTransparency и в обратную сторону.

void **handleEvent**(*const* SDL_Event &**event**);	Метод для определения нажатия кнопки.
void Button::**setEnabled**(bool **enable**)	При enable == true устанавливается максимальная прозрачность, при enable == false устанавливается минимальная прозрачность и отключается анимация мигания

Cell	
Cell(SDL_Renderer *__renderer__, *const* std::string &__file__, *const* SDL_Rect &__currentRect__, *const* SDL_Rect &__borders__);	Конструктор для создания ячейки в барабане, принимающий: - рендер - файл текстуры - изначальное положение ячейки - границы области, внутри которой она может перемещаться
void **render**();	Метод отрисовки ячейки барабана. Если установлен флаг isSplitted, то отрисовывается также “дополнительная” текстура fadeTexture, представляющая собой появляющуюся часть ячейки сверху.
void **move**(float **step**);	Метод для перемещения ячейки по вертикали. Проверяет выход за границы ячейки и устанавливает соответствующие размеры ячейки при их пересечении. Перемещение наверх не поддерживается.
void **reset**(int **y**);	Сбрасывает размеры основной текстуры до изначальных и помещает ячейку по вертикальной оси в положение ‘y’

Drum	
Drum(SDL_Renderer *__renderer__, *const* std::vector<std::string> &__files__, *const* SDL_Point &__pos__, *const* SDL_Point &__cellSize__);	Конструктор для создания одного барабана, принимающий: - рендер - вектора файлов текстур для каждой ячейки - положение барабана - размер ячеек

void **render**(float timeStep);	Метод отрисовки ячеек барабана. При “вращении” перемещает все ячейки на шаг moveStep. При окончании перемещения (когда заканчивается “дистанция перемещения” targetShufflingDistance, останавливает вращение и фиксирует положение ячеек.
void **startShuffle**(float seconds);	Запускает вращение барабана. Генерирует случайные результирующие индексы положения ячеек и скорость вращения, на вход принимает количество секунд, которое необходимо вращать барабан.

FPSCounter	
FPSCounter(SDL_Renderer *renderer, const std::string &fontFile, SDL_Rect pos, SDL_Color color);	Конструктор для объекта, отображающего количество кадров в секунду и принимающий: - рендер - файл шрифта - положение - цвет текста
void **render**(float fps);	Отображает значение fps

Texture	
static std::unique_ptr<Texture> **loadTexture**(SDL_Renderer *renderer, const std::string &file, SDL_BlendMode blendMode = SDL_BLENDMODE_BLEND);	Фабричный метод для загрузки текстуры, принимающий: - рендер - файл текстуры - режим текстуры
static std::unique_ptr<Texture> **loadTextTexture**(SDL_Renderer *renderer, const std::string &text, TTF_Font *font, SDL_Color color);	Фабричный метод для загрузки текстовой текстуры, принимающий: - рендер - текст для отрисовки - шрифт - цвет
void renderTexture(const SDL_Rect &dst);	Метод для отрисовки текстуры в положение dst

void **renderTexture**(*const* SDL_Rect &**src**, *const* SDL_Rect &**dst**);	Метод для отрисовки части текстуры src в положение dst
void **updateTransparency**(Uint8 **a**);	Метод для изменения степени прозрачности текстуры

TextureLoader	
static SDL_Texture* **loadBMP**(SDL_Renderer *__renderer__, *const* std::string &__file__)	Фабричный метод для загрузки текстуры из .bmp файла, принимающий: - рендер - файл текстуры
static SDL_Texture* **loadTextTexture**(SDL_Renderer *__renderer__, *const* std::string &__text__, TTF_Font *__font__, SDL_Color __color__)	Фабричный метод для создания текстовой структуры, принимающий: - рендер - текст для отображения - шрифт - цвет текста

FPSTimer	
FrameInfo **getFrameInfo**();	Метод для подсчета количества кадров в секунду и времени кадра. Необходимо вызывать при каждой итерации цикла отрисовки.

UniformRandom	
static **int** **uniformInt(int min, int max);**	Метод для генерации числа от min до max на основе равномерного распределения

Улучшения & недостатки

- Не учитываются размеры текстуры и требуется размер 100x100 или менять в исходном коде, иначе ячейки “плывут”
- FPSTimer выделить в класс-одиночку для использования единого инстанса во всех участках программы
- Отсутствие проверок на неадекватные значения количества ячеек и т. д.
- Вместо исключения при каких-либо ошибках не будут отрисовываться объекты, которые не смогли загрузиться без информации об этом, кроме логов -> MessageBox

- Классы для работы с текстурами представляют только минимум функций, который нужны по задания
- Поддерживается только .bmp из коробки SDL2, SDL2_img не подключался из-за отсутствия необходимости
- Отсутствует ручное ограничение частоты кадров - для этого установлен флаг *SDL_RENDERER_PRESENTVSYNC* , синхронизирующий принудительно частоту с монитором
- Не корректируется размер области с текстом о количестве кадров в секунду в зависимости от количества цифр
- Местами может хромать форматирование кода из-за не до конца настроенного автоформатирования
- Сборка на Linux может потребовать корректировки CMakeLists.txt