**DEV-Dokumentáció**
Tánczos János - SNAKE
GWVABC

**File tree:**
Main.c
Lib/
    Game/
        game.c - game.h
        snake.c - snake.h
    Menu/
        menu.c - menu.h
        button.c - button.h
        scoreboard.c -scoreboard.h
    rendering.c - rendering.h
    save_system.c -save_system.h

**game.c - game.h**
```
/// Full game loop, handles everything from rendering and game logic
/// @param renderer [in] constraints the SDL_renderer
void StartGame(GameRenderer *renderer);
```

**button.c - button.h**
```
/// Constraints the boundary of the button
typedef struct Button {
   SDL_Rect boundary;
} Button;

/// Creates a button on the scene
/// @param gameRenderer is the renderer
/// @param texture SDL_Texture of an image
/// @param pos the position of the button (center)
/// @return a Button it self
Button RenderButton(GameRenderer *gameRenderer, SDL_Texture *texture, Vector2 pos);

/// Detect the overlap of the given Button and a Vector2
/// @param button
/// @param cursor
/// @returns true if there is overlap and false if there in not
bool DetectOverlap(Button *button, Vector2 cursor);
```

**menu.c - menu.h**
```
/// Opens the main menu and renders every related things
/// @param GameRenderer
/// @returns WindowState enum what describes what state does the user selected from
the menu
enum WindowState OpenMenu(GameRenderer *renderer);
```

**scoreboard.c -scoreboard.h**

```c
///Renders the score board on top of the menu
///@param renderer the Gamerenderer
///@returns void
void ShowScoreboard(GameRenderer *renderer);
```

**snake.c - snake.h**

```c
/// Direction
enum Direction {
   UP,
   DOWN,
   LEFT,
   RIGHT
};
/// A linked list as the snake
typedef struct Snake {
   Vector2 bodyPart;
   struct Snake *next;
} Snake;

/// Creates the snake from scratch
/// @param startPos the coordinates of the whole Snake
/// @param length the initialization length if the Snake
/// @returns A Snake struct what it self is a linked list
/// @attention You have to Free the snake with the FreeSnake function
Snake *CreateSnake(Vector2 startPos, int length);

/// Moves the snake to the given direction
/// @param snake the snake is Self
/// @param nextDirection direction of the move
bool MoveSnake(Snake *snake, enum Direction next);

/// Returns the last body part's postion
/// @param snake
/// @returns true if the move can be done, and false if something is blocking the
snake it self
Vector2 LastSnakeBody(Snake *snake);

/// Frees the snake
/// @param snake
void FreeSnake(Snake *snake);

/// Ads an elemt to the snake, (it can be used to create a snake it self)
Snake *AddElementToSnake(Snake *snake, Vector2 v);

/// Expands the snake to the given direction
/// @param snake the snake is Self
/// @param nextDirection direction of the expansion
void ExpandSnake(Snake *snake, Vector2 last);
```

**rendering.c - rendering.h**

```
/// States of the app window
enum WindowState {
   GAME,
   MENU,
   SCORE_BOARD,
   EXIt
};
/// Constraints everything what essential info tu rendering
typedef struct GameRenderer {
   SDL_Renderer *renderer;
   enum WindowState state;
} GameRenderer;

/// A 2D vector with X and Y coordinates
typedef struct Vector2 {
   int x, y;
} Vector2;

/// Initialize the renderer this step makes the program graphical
/// @return a GameRenderer object what used in every other rendering specific task
GameRenderer InitGameRenderer();

/// Creates a "pop-up" like dialog, with a yes or no question
/// @param renderer the GameRenderer
/// @param question a char array
/// @returns the answer of the Yes-no question
bool CreatePopUp(GameRenderer *renderer, char question[]);

/// Creates a "pop-up" like dialog, with an input field
/// @param renderer the GameRenderer
/// @param title a char array what will be the title of the dialog
/// @param subTitle a smaller title  for more info
char *CreateInputPopUp(GameRenderer *renderer, char title[], char subTitle[]);

/// Loads a font from the disk
/// @returns TTF_Font pointer
TTF_Font *LoadFont();

/// Gets the size of a texture
/// @param texture
/// @returns a Vector2 with the dimensions
Vector2 GetTextureSize(SDL_Texture *texture);

/// Renders the text to the screen
/// @param renderer
/// @param font the loaded font
/// @param txt the text to be rendered
/// @param pos the position of the text
/// @param size the scale of the text "1" is the normal
void RenderText(SDL_Renderer *renderer, TTF_Font *font, char txt[], Vector2 pos, float
size);
```

```
/// Copied from infoC
/// Generates an input text field
/// @param dest the destination of the result
/// @param hossz it must be smaller than the lenght of the destination
/// @param teglalap what surrounds the  input-field
/// @param hatter the color of the background
/// @param szoveg the color of the txt
/// @param font the font of the text
/// @param renderer
bool input_text(char *dest, size_t hossz, SDL_Rect teglalap, SDL_Color hatter,
SDL_Color szoveg, TTF_Font *font,
               SDL_Renderer *renderer);
```

**save_system.c -save_system.h**

```
/// A package for the snake to help loading/saving
typedef struct SnakeData {
   Snake *snake;
   enum Direction direction;
} SnakeData;

/// Score struct
typedef struct Score {
   char *name;
   int value;
} Score;

/// A linked list from Score structs, represents a package what will be saved
typedef struct ScoreList {
   Score score;
   struct ScoreList *next;
} ScoreList;


/// Saves the snake'c current state to the disk
/// @param snake a Snake it self
bool SaveSnake(SnakeData snakeData);

/// Loads the snake from the fs
/// @returns a "Snake" type linked list
SnakeData LoadSnake();

/// frees the ScoreList
/// @param scoreList what you wanna set free
void FreeScoreList(ScoreList *scoreList);

/// Save the score to the disk
/// @param score
/// @returns true if succeeded and false if not
bool SaveScore(Score score);

/// Loads the score from the disk
```

```c
/// @returns a linked list with the scores
/// @attention you have to free the returned list after use, with FreeScoreList()
ScoreList *LoadScore();
```