Anim.c:

Animation CreateAnim(const char* sprite_sheet_path, int animFrames, int frameDelay, Vector2 size);

brief; Creates an animation to play on a rendered screen.
Param: sprite_sheet_path The path directed to the sprite's texture (in resources folder)
param: animFrames The animation frames
Param: frameDelay The animation frame-change delay
Param: size The animation's size
Return: An Animation object with the essential information to emulate an animation

void UpdateAnim(Animation* anim);

Brief: Updates an animation frame to the next scheduled frame.
Param: anim A pointer to the to-be-updated Animation object

void StartAnim(Animation* anim, Vector3 pos);

Brief: Spawns an animation.
Param: anim A pointer to the to-be-spawned Animation object
Param: pos The spawn position (3D Vector)

void DrawAnim(Animation anim, Camera cam);

Brief: Plays an animation, if it's scheduled to be played.
Param: anim The to-be-played Animation object
Param: cam The Camera object in which the animation will be played

Game.c:

```
void FixShipPosition()
```

Brief: A bug-fixing procedure
Details: This procedure fixes a bug that occurred when trying to spawn each ship in a rendering screen. Occasionally, the ship's Y position coordinate would "displace" itself to a lower point (between 10-20 points off). Therefore, this procedure forces ships to ALWAYS spawn in the Y position coordinate which responds to their type. This bug is reproducible by removing the call in DisplayRealTimeGameScreen and DisplayTurnBasedGameScreen procedures.

```
void DisplayRealTimeGameScreen(Ship_data ship_data, Obstacles obstacles, const Model* game_models, const Sound* game_sounds, Texture2D* game_textures, Animation* anim_list, const Texture2D* water_textures);
```

Brief: Displays the real-time gameplay screen and manages the game.
Param: ship_data Holds the spawned ships' data
Param: obstacles Holds the spawned islands and rocks' data
Param: game_models Holds the models to be rendered in-game
Param: game_sounds Holds the sounds to be played in-game
Param: game_textures Holds the textures to be drawn in-game
Param: anim_list Holds the animations to be played in-game

```
Param: water_textures Holds the water textures for the water animation to be
played in-game

void DisplayTurnBasedGameScreen(Ship_data ship_data, Obstacles obstacles,
const Model* game_models, const Sound* game_sounds, Texture2D*
game_textures, Animation* anim_list, const Texture2D* water_textures);
Brief: Displays the turn-based gameplay screen and manages the game.
Param: ship_data Holds the spawned ships' data
Param: obstacles Holds the spawned islands and rocks' data
Param: game_models Holds the models to be rendered in-game
Param: game_sounds Holds the sounds to be played in-game
Param: game_textures Holds the textures to be drawn in-game
Param: anim_list Holds the animations to be played in-game
Param: water_textures Holds the water textures for the water animation to be
played in-game

void DrawGameState(Ship_data ship_data, Camera camera, RenderTexture
screenShip, Obstacles obstacles, char real_or_turn,
const Model* game_models, Ship current_player_ship, Texture2D*
game_textures, const Animation* anim_list, const Texture2D* water_textures);
Brief: Schedules the textures, models and animations to be displayed in any
gamemode and updated to camera
Param: ship_data Holds the spawned ships' data
Param: camera The Camera object in which everything is displayed and updated
Param: screenShip The screen in which textures will be drawn
Param: obstacles Holds the spawned islands and rocks' data
Param: real_or_turn Determines if the gamemode is real-time or turn-based
Param: game_models Holds the models to be rendered in-game
Param: current_player_ship The screen ship's instance
Param: game_textures Holds the textures to be drawn in-game
Param: anim_list Holds the animations to be played in-game
Param: water_textures Holds the water textures for the water animation to be
played in-game

void DrawUI(Ship current_player_ship, const Texture2D* game_textures,
RenderTexture screenShip);
Brief: Part of the DrawGameState procedure
Details: Specifically, draws UI-related elements, such as the FPS text, the
power and reload bars
Param: current_player_ship The screen ship's instance
Param: game_textures Holds the textures to be drawn in-game
Param: screenShip The screen in which textures will be drawn

void UpdateVariables(Ship_data ship_data, Sound explosion, Obstacles
obstacles, Animation* explosion_anim);
Brief: A procedure which checks for changes and updates the ships' state
(attributes)
Param: ship_data Holds the spawned ships' data
Param: explosion The explosion sound to be played in-game
Param: obstacles Holds the spawned islands and rocks' data
Param: explosion_anim The explosion animation to be played in-game
```

```
void *DecreaseTime(void *arg);
Brief: Decreases a time variable by 1 per second
Details: Used in threads for every gamemode
Note: The function is NOT defined to return a pointer, please ignore any
warnings.
Param: arg The time variable to decrease. This argument is passed through
the pthread_create function
Return: Nothing, according to its usage in DisplayRealTimeGameScreen and
DisplayTurnBasedGameScreen

void *DecreaseCounter(void *arg);
Brief: Decreases a counter variable by 1 per second
Details: Used in threads for every gamemode
Note: The function is NOT defined to return a pointer, please ignore any
warnings.
Param: arg The counter variable to decrease. This argument is passed through
the pthread_create function
Return: Nothing, according to its usage in DisplayRealTimeGameScreen and
DisplayTurnBasedGameScreen


main.c:
int main();
Main contains initialization of many variables passed down to other
functions, the basic game loop, and program de-initialization before process
termination (freeing up resources).

Obstacles.c:
Island CreateIsland(Texture2D sand_tex, Model toppings, Vector2
corner_bound, Vector2 opp_corner_bound);
Brief: Defines an Island object.
Param: sand_tex The island's sand texture
Param: palm_tree The island's palm tree model
Param: corner_bound The corner of the game bounds, used in spawning (2D)
Param: opp_corner_bound The opposite corner of corner_bound, used in
spawning (2D)
Return: The created Island object

Island* CreateAllIslands(Texture2D sand_tex, Model toppings, Vector2
corner_bound, Vector2 opp_corner_bound, int island_count);
Brief: Defines an array of Islands and returns it.
Param: sand_tex The islands' sand texture
Param: toppings The islands' palm tree model
Param: corner_bound The corner of the game bounds, used in spawning (2D)
Param: opp_corner_bound The opposite corner of corner_bound, used in
spawning (2D)
Param: island_count The number of islands to generate
Return: An array of Islands created
```

```
Rock CreateRock(Texture2D rock_tex, Vector2 corner_bound, Vector2
opp_corner_bound);
Brief: Defines a Rock object.
Param: rock_tex The rock's texture
Param: corner_bound The corner of the game bounds, used in spawning (2D)
Param: opp_corner_bound The opposite corner of corner_bound, used in
spawning (2D)
Return: The created Rock object

Rock* CreateAllRocks(Texture2D rock_tex, Vector2 corner_bound, Vector2
opp_corner_bound, int rock_count);
Brief: Defines an array of Rocks and returns it.
Param: rock_tex The rocks' texture
Param: corner_bound The corner of the game bounds, used in spawning (2D)
Param: opp_corner_bound The opposite corner of corner_bound, used in
spawning (2D)
Param: rock_count The number of rocks to generate
Return: An array of Rocks created

Obstacles CreateObstactlesInstance(Island* island_list, int island_count,
Rock* rock_list, int rock_count);
Brief: Creates an Obstacles object, in which are stored the data for any
obstacle spawned in-game.
Param: island_list The array of Islands generated
Param: island_count The number of Islands generated
Param: rock_list The array of Rocks generated
Param: rock_count The number of Rocks generated
Return: The generated Obstacles instance

Obstacles init_obs(Texture2D sand_tex, Texture2D rock_tex, Model palm_tree);
Brief: A procedure which creates the required parameters to generate an
Obstacles object.
Param: sand_tex The islands' sand texture
Param: rock_tex The rocks' texture
Param: palm_tree The islands' palm tree model
Return: The final Obstacles instance


screens.c:
void InitMainWindow();
Brief: Initializes the game's window.

void DeinitMainWindow();
Brief: De-initializes the Main window on exit.

void DisplayMainScreen(Sound click, Obstacles *obstacles, Texture2D
sand_tex, Model palm_tree, Texture2D rock_tex);
Brief: Displays the game's Main screen.
Param: click The button click sound
```

```
Param: obstacles A pointer to the generated Obstacles instance
Param: sand_tex The islands' sand texture
Param: palm_tree The islands' palm tree texture
Param: rock_tex The rocks' texture

void DisplayGamemodesScreen(Sound click, int* player_count_addr, char*
real_or_turn_addr);
Brief: Displays the gamemode selection screen.
Param: click The button click sound
Param: player_count_addr A pointer to the player count integer
Param: real_or_turn_addr A pointer to the variable declaring if the gamemode
is real-time or turn-based

void DisplayShipSelectScreen(Sound click, int* type_list, int player_count,
char real_or_turn);
Brief: Displays the ship selection screen.
Param: click The button click sound
Param: type_list An array declaring each player's ship type
Param: player_count The amount of players participating in the game
Param: real_or_turn The variable which declares if the game is real-time or
turn-based

void DisplayTeamSelectScreen(Sound click, int* team_list, int player_count,
char real_or_turn);
Brief: Displays the team selection screen.
Param: click The button click sound
Param: team_list A pointer to the array declaring each player's team
Param: player_count The amount of players participating in the game
Param: real_or_turn The variable which declares if the game is real-time or
turn-based

void DisplayGameOverScreen(char* wintext, Sound click);
Brief: Displays the game over screen, when the game ends.
Param: wintext The text which declares the winner of the game
Param: click The button click sound

void DisplayOptionsScreen(Sound click, bool* bgm_en);
Brief: Displays the options (settings) screen.
Param: click The button click sound
Param: bgm_en Declares if background game music is enabled or not

void DisplayControlsScreen(Sound click);
Brief: Displays the control settings screen.
Param: click The button click sound

void DisplayAboutScreen(Sound click);
Brief: Displays the about screen, with credits and gameplay instructions.
Param: click The button click sound

void DisplayGameMenuScreen(Sound click, Obstacles obstacles);
```

```
Brief: Displays the game menu screen, when pressing the ESCAPE button in-
game.
Param: click The button click sound
Param: obstacles The Obstacles instance
```

ship.c:

```
Ship* SetupShips(int player_count, const int* type_list, const int*
team_list, Obstacles obs, Model* ship_models);
Brief: Sets up the appropriate amount of ships for a game to begin.
Param: player_count The amount of Players (ships) to be spawned
Param: type_list An array of integers declaring each ship's type
Param: team_list An array of integers declaring each ship's team
Param: obs The Obstacles object, holding data for spawned obstacles
Param: ship_models An array of Models with the ships' models
Return: A pointer to the generated Ship object

Ship_data CreateShipData(int player_count, int* type_list, int* team_list,
Obstacles obs, Model* ship_models);
Brief: Creates a Ship_data structure, which holds essential information
about ships, their type and team indexes.
Param: player_count The amount of players (ships) to be generated
Param: type_list An array of integers declaring each ship's type
Param: team_list An array of integers declaring each ship's team
Param: obs The Obstacles object, holding data for spawned obstacles
Param: ship_models An array of Models with the ships' models
Return: The generated Ship_data structure

Ship LoadShip(int type, const cJSON *shipState, int playercount);
Brief: Loads a ship from a saved game state properly.
Param: type The ship's type
Param: shipState A pointer to the cJSON array, which holds its saved state
Param: playercount The amount of players who started the game (derived from
the game state)
Return: A Ship object, properly set up to continue the game

void *EndGame();
Brief: Executes commands to properly finish the game.
Details: This specific function is used in a pthread, in order to NOT
interrupt the main game control. It waits for a
second (1000000 microseconds) and changes the screen to the game over, to
announce the winner.
Note: The function is NOT defined to return a pointer, please ignore any
warnings.
Return: Nothing, according to its usage in CheckWin procedure

void CheckWin(Ship_data ship_data);
Brief: Responsible to check who the winner of a game is (either team or
solo)
Param: ship_data The Ship_data object, containing data for the ships
```

```
int FindNextAliveShipIndex(Ship_data ship_data, int start_index);
Brief: Calculates the next alive ship's index number inside the ship_data
structure
Param: ship_data The Ship_data object
Param: start_index The starting index to begin the calculations
Return: The calculated index (integer)

void CheckMovement(Ship *ship, Sound fire);
Brief: Responsible to CONSTANTLY check for any button press and apply the
change in ships' movement and control.
Param: ship A pointer to the Ship object required to control
Param: fire The fire sound to be played in-game

void InitializeCannonball(Ship* ship);
Brief: Initializes the cannonball for the Ship object to use in-game.
Param: ship A pointer to the Ship object to initialize its cannonball

void UpdateCannonballState(Cannonball* cannonball, Sound splash, Animation*
splash_anim);
Brief: Updates the given cannonball's firing state.
Param: cannonball The Cannonball object to update
Param: splash The splash sound to be played in-game, when the cannonball
reaches the water
Param: splash_anim The splash animation to be played in-game, when the
cannonball reaches the water

void UpdateShipCamera(const Ship *ship, bool first_person);
Brief: Updates the given Ship's camera to adjust to any movement changes.
Param: ship A pointer to the Ship to update
Param: first_person Declares if the camera is in first or third person

void CheckHit(Ship* player_ship, Ship* enemy_ship, Sound explosion,
Obstacles obstacles, Animation* explosion_anim);
Brief: Checks for any kind of interaction between hitboxes (ships,
cannonballs).
Param: player_ship The current Ship playing
Param: enemy_ship The enemy Ship
Param: explosion The explosion sound to be played in-game when an
interaction occurs
Param: obstacles The Obstacles object, holding data for spawned obstacles
Param: explosion_anim The explosion animation to be played in-game when an
interaction occurs

void CheckCollisionWithBounds(Ship *ship, BoundingBox bound);
Brief: CONSTANTLY checks if the given Ship collides with the game boundaries
(SkyBox's scaled Bounding Box).
Param: ship A pointer to the Ship to check for collision
Param: bound The game boundaries
```

util.c:

```
bool strtobool(const char *input);
Brief: Converts a string to bool (default to false)
Param: input The string to be converted
Return: The appropriate Boolean value

char *booltostr(bool input);
Brief: Converts a bool to string (default to "false")
Param: input The bool to be converted
Return: The appropriate string value

void AddScreenChangeBtn(Rectangle rec, const char* text, Vector2
mouse_point, Sound click, screen* current_screen, screen scr, bool sfx_en);
Brief: Adds a working button.
Param: rec The button's rectangle
Param: text The button's text
Param: mouse_point The current mouse location on-screen
Param: click The button click sound
Param: current_screen A pointer to the current screen variable
Param: scr The screen to switch to when the button is pressed
Param: sfx_en Declares if the sound effects are enabled or not (derived from
settings)

void AddSetting(bool* setting, const char* setting_name, Rectangle rec,
Sound click, bool sfx_en);
Brief: Adds a setting in the Options screen.
Param: setting A pointer to the setting
Param: setting_name The setting's name
Param: rec The setting's rectangle
Param: click The button click sound
Param: sfx_en Declares if the sound effects are enabled or not

void AddButtonSetting(int *key, Rectangle rec, const char *label_name, int
btn_id);
Brief: Adds a button setting in the Controls screen.
Param: key The key to control
Param: rec The button's rectangle
Param: label_name The key's functionality label
Param: btn_id The button's id number


void LoadSettings(bool* bgm_en);
Brief: Loads settings, parsed from config.ini.
Param: bgm_en Declares if the background game music is enabled or not

void UpdateSettingsConfig(setting settings);
Brief: Dynamically parses any change in settings during runtime to
config.ini.
Param: settings The settings object
```

```
static int parseHandler(void* user, const char* section, const char* name,
const char* value);
Brief: The main INI parser, parsing config.ini and setting up the settings
object (follows prototype structure, check reference).
Param: user The object to be set up
Param: section The current INI section
Param: name The current INI item's name
Param: value The item's value
Return: Returns 1 in success, 0 in failure

cJSON *create_ship_json(Ship ship, int type);
Brief: Creates a JSON array, storing essential information about a ship's
state.
Param: ship The ship to be saved
Param: type The ship's type
Return: A pointer to a cJSON array object

Color ReturnColorFromTeamInt(int col_int);
Brief: Returns the color of each team, depending on its id.
Param: col_int The team's id
Return: The appropriate Color code

void SaveGameState(Obstacles obstacles);
Brief: Saves the current game state, parsing the data to a JSON file.
Param: obstacles The Obstacles instance, holding the essential information
about each and every spawned obstacle in the game

int LoadGameState(Obstacles *obstacles, Ship_data *ship_data, Texture2D
sand_tex, Model palm_tree, Texture2D rock_tex);
Brief: Loads a game state from a JSON file, parsing its data and
initializing the appropriate variables for smooth gameplay.
Param: obstacles A pointer to the Obstacles object
Param: ship_data A pointer to the array holding the ships' data
Param: sand_tex The islands' sand texture
Param: palm_tree The islands' palm tree model
Param: rock_tex The rocks' texture
Return: Returns 1 in success, 0 in failure

void CheckFullscreenToggle();
Brief: Toggles fullscreen and updates settings dynamically, when F11 is
pressed anytime during runtime.
```