Basic Engine Documentary

1. Build a basic engine to realize a variation of the classic game Asteroids
    - You can use the provided assets or other sprites/sprite sheets
    - You can also add animations

We used only provided assets and self drawn start screen
Animations not implemented

2. Game engine architecture. (6 points)
    - Implement Game class that coordinates initialization, update and shutdown and any implemented subsystems (see Game class from exercise).

We used Game Class from exercise - find in Game.h/cpp

- Implement GameStateManager and States (see slides Game Engine Architecture).

GameStateManager - our GameStateManager is our Game.h/cpp
All states (gamestate, menustate) implement State class - find in folder states
class Gamestate.h/cpp, MenuState.h/cpp, EndState.h/cpp
Game Class is also responsible for delegating our KeyPress Events to the InputManager

- Implement InputManager which translates SFML events into internal events, i.e., to query abstract bindings like "move_left" (see slides Input).

Inputmanager - find in InputManager.h/cpp
Implemented as Singleton with Custom Player Datastructure which can be instantiated for virtual Player Input. See InputManager::AddPlayer()

- Implement a component-based game object and appropriate components that encapsulate functionality (see slides Game Engine Architecture)

GameObject - find in GameObject.h/cpp, gameobject inherits Transformable
All added Components on a Gameobject are stored in -
std::vector<std::shared_ptr<Component> mComponents;
All Components - find in folder Components, all derived components classes implement class component.h/cpp
Our Component Lifecycle Methods are managed by the GameObject class
- see GameObject::Update()
New GameObjects are managed by the GameObjectManager.h/cpp(implemented as Singleton)

- In the end you create a GameObject for background (astroids_bg.jpg), one for each player (sprite ship.png) and one for the camera.

Instantiations find in GameState::init()
We created a separate Component for our Background Background.h/cpp, for our Camera FollowPlayer.h/cpp & Camera.h/cpp and for the ship Ship.h/cpp

- Gracefully shutdown game and free the memory (no exit(0)).

See Game::Shutdown()
The window can also be closed in Game::Update() by pressing the Escape key

3. Transition between start screen and game. (1 point)
   - Use the GameStateManager and two State implementations.
   - Trigger transition by pressing "space".

Used States – in folder States class GameState.h/cpp & MenutState.h/cpp
Trigger Transition – find in MenuState::Update()
(EndState.h/cpp tba)

4. Background game object. (1 point)
   - Render a background (ideally using a RenderComponent interface implementation).
   - There is no depth testing, so render the background before the rest.
   - Note that the camera moves over this background, i.e., the background is not rigidly attached to the camera (see sf::View of SFML for rendering the camera view).

Our Background image is a GameObject and gets the Component Background.h/cpp
Background Class implements SpriteRenderer Class, which implements Component Class
Background Component sets Texture to FullWindow on start() – found in Background.cpp

5. Two player characters moving around. (3 points)
   - Control players using up for going into the current direction, use left/right to rotate players so that they change directions
   - Mappings player one: keyboard left, right, up
   - Mappings player two: keyboard w, a, d
   - Players are ships and are rendered as a sprite using e.g., ship.png
   - The players must not leave the playing field and start at the center of the background. Playing field is given by dimensions of the asteroids_bg.jpg
   - The transparency value of the ship sprite is 255,128,255

Ship Game Components creates a new virtual Player input via the InputManager.
You can dynamically remap the keybindings of the corresponding GameObject
 – see Ship.h/cpp
TransparencyMask is applied in SpriteRenderer.h/cpp

6. The camera moves with one of the players. (2 points)
   - Zoom the view, so that only small part of the playing field is visible
   - Rigidly attach the view to the ship
   - Move the view with the ship

The camera gameobject gets the components FollowPlayer.h/cpp and camera.h/cpp which holds the logic for zooming and setting the view on a specific player
Change camera view from player1 to player2 with pressing V while in GameScreen

Bonus (3 points):
1. When the ship reaches a border it appears at the opposite side of the playing field (adjust position of game object directly; calculate position based on bounds of background)
2. Add a switch (Key: ".") to toggle between players switching screen position when reaching the border (bonus behavior) and players not being able to leave the playing field (standard behavior).
3. Output the current border crossing/blocking behavior to the console.

Switch Key is implemented in GameState.h/cpp
It is implemented as a public static Boolean Method for easy access in every Component – Output gets printed via Console
Ship Component uses public static Boolean Method for Behavior at Borders

Additional Notes
Hourglass.h/cpp is a Singleton for retrieving the DeltaTime in every Class