Speedway:

Overview:

The framework used is OpenGL, based on the Framework provided by the course.

Game logic and physics are handled over Physx4.1 while rendering is done in OpenGL.

1920x1080 screen size is default, but can be changed through ini file.

Detailed Classes Explanation:

Main:

Methods:

The main method intitalizes OpenGL and PhysX, loads textures and models and creates the objects for the main game loop. Every visible PhysX object is rendered as a OpenGL object. After initializing OpenGL and window, a loading screen is presented until all other objects are loaded.

For the player character, PxCapsuleController is used. User inputs are handled using mouse button and key callback functions, setting different boolean variables to true on press or false on release to allow for multi key input at the same time.

Those booleans are used in the updateCharacterMovement() and updateCharacterSize() functions. When a direction (forward, backward, left, right) is true, velocity is added in this direction in relation to the current camera position (returned by the update function of the camera).
isGrounded() is used to check if the character is currently moving on the y axis, and sets the boolean deciding if we are already jumping. If not jumping, the jump key adds velocity to the y direction, with negative velocity being added every frame that is jumping.
When a character turns smaller or bigger, we check if they are already min or max size, and if not change the size of the controller and corresponding Geometry object.
The current size also affects velocity at a 50% rate ((1+curren.tSize)/2).

Controls:
AWSD are used for x and y axis directions
Space for jump
Q to turn water in ice
RMB and LMB to turn bigger and smaller
Mouse movement to change view direction
ESC to quit
V for debug mode flying

Main Loop:

In the main loop, every loop a new deltatime is measured. If deltatime < 1/60 (desired fps) the game logic loop is skipped.
In the game logic loop, character size and movement are updated and applied, the scene is simulated, and the OpenGL objects transformed accordingly. The playerCamera position is updated, and returns our ForwardVector used for leftVector and movement. Events and CursorPosition are polled outside of the game loop and Objects rendered every frame regardless of game logic skipping frames.

GamePlay:

Reach the goal found on the platform above past the waves and big block as fast as possible (triggers win screen). The movement inputs are always relative to camera position (forward = forward).
Water can be turned into walkable ice by holding the “Q” button while entering the trigger shape of the water.
There is a game over trigger under the map. It is currently found by touching the outside of the big plattform at the start (triggers game over screen) or walking into the water without turning it to ice.
By becoming larger, the players can move quicker and jump further, but can use turning smaller to get past platforms (like the big block blocking the path).

PlayerCamera:

PlayerCamera is used for both the _viewProjectionMatrix and _cameraDirection using std::vec3 and glm::mat4. It is created with a starting position, field of view and a boolean to use first or third person, since we want to find out which one fits the playstyle better.

On update, the x and y mouse offset are calculated and used to create a direction vector from the player characters position (newPosition). The y axis is used for the camera, but ignored for the returned forwardVector. The other variables are also calculated in the update and then queried with getters, e.g. viewProjectionMatrix for shaders and Yaw for character rotation matrix.

changeSize() changes the field of view to simulate a change in size of the playerCharacter.

Camera

Taken from ECG and only used for debugging.

Shader2d, Texture2d, SpriteRenderer

Based on https://learnopengl.com/In-Practice/2D-Game/Setting-up with adjustments to be used for rendering loading, win and lose screens in 2d and fixed image colors. Uses stb_image library.
Also used by our Implementation of LensFlare and Particle Systems

Geometry:

Taken from ECG for a base of simple Shapes and adjusted with scale&rotate methods, as well as own methods for more complicated Shapes. Scale and rotation for the object of a room and model of the main character was made with blender.

IniReader, Material, Shader, Texture, Light, Utils:

From ECG. Where applicable for our gameplay and graphics goals, changed or replaced by own implementations.

Level:
Level was build with blender

Platforms:
System of moving platforms and their simple animation.

Scripting Language Integration:
Using general LUA script to alter on a fly speed/jump of character. Behavior/movement of trap. Speed of a platforms
Tutorial:
https://www.youtube.com/watch?v=4l5HdmPoynw

Heads-Up Display:
Using **SpriteRenderer** to draw HUD. Non-trivial HUD can be drawn with **discard** on the side of fragment shader. F4 - switch off/on

WaterIce:

Turning of water into ice was implemented as static rigid body. Initially the rigid body is outside of a space of water and the main character can fall in a trigger space of gameover. When the character switch water to ice - we can use an ice platform to pass water

Model and loading of an object:

Model is loaded with tinyobjloader library from Vulkan tutorial. It can load vertices, indices and uv-coordinates for one texture: https://vulkan-tutorial.com/Loading_models
Model from this tutorial was loaded as starting area: viking_room.obj, viking_room.png
Vertices and indexes are used for cooking of PhysX rigid static bodies as well.

For character was used free to use model of tibetan fox with one texture:

https://open3dmodel.com/3d-models/tibetan-fox_482026.html

For trigger of the ending of a game for this prototype was used free model of a trophy

FlareManager:

Based on the tutorial from https://www.youtube.com/watch?v=OiMRdkhvwqg which is made in Java, we converted it to C++ and adjusted it with our file structure. When the sun object is in the screens field of view, an array of lens flare textures with alpha values is rendered on a vector from screen center to sun position.

ParticleGenerator:

Based on this tutorial:
http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/

but made into a callable class with methods and using our existing Rendering, Texture and Shader classes. Creates a colourful smoke like effect that should look like an aurora borealis. 2D textures are rotated with playercamera direction inputs to create this effect while saving processing time, since the implementation is done GPU side.

Baked lightning:
For static objects that have baked lightning, a new object with new uv-maps was created with corresponding diffuse texture and lightmaps(direct+indirect light) with the same resolution. In leaves.frag shader these 2 textures are mixed. The same operation was made for every object with baked lightning as joining all objects of a scene could cause unexpected uv-mapping errors.
Based on tutorial:
https://youtu.be/VS4rqkqmg7Y
https://youtu.be/Y1yrGxJ9g3M

Video textures:
Two textures for a plane were created. These textures are updating every frame. Video for this texture was converted in 24 fps and after with VLC player frames of this video were saved as images and converted into .dds format for a texture of material.

Vertex shader animation:
A Sub-divided plane(water) or sphere(lava) with a big amount of vertices is used to make a small distortion of points on a side of the vertex shader water.frag with periodic functions(wave effect). Uniform of time is used to create animation.
Based on tutorial:
https://youtu.be/5yhDb9dzJ58

Simple normal mapping:
Normal map was sent as uniform in fragment shader to make illumination of point light of an object with a baked texture. Point light is moving.
Tutorial:
https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Additional Libraries used:

tiny_obj_loader, PxPhysicsAPI

Known Issues/Work in Progress:

- Ice/Water interaction with button doesnt work if already touching object, will be fixed
- Character Model is visually smaller than character controller object. All interactions are tested with controller object, visual object will be adjusted
- Jump doesnt work if running downwards objects, will try to fix this one
- Moving Platforms currently in main with fixed values, will be moved to external class after more pressing issues are fixed
- Possibly not all memory allocations are released at the end of game, working on it