Documentation of
Polar Adventures

Overview

Polar Adventures is a game that mixes platforming, riddle solving and adventuring. This leads to an experience one could describe as a 3D variant of an old school Zelda game. Upon start the player is placed on the top side of a cube, which is representing the game world. The objective of the game is to find five snowballs, that are spread around the cube, collect them and bring them back to the top side of the cube, where they can be dispensed to the collection point.
These snow balls, however, shrink with time passing (melting). The earlier the snowballs are collected the more points they will yield, when thrown into the collection point. Should the player take too long to find the snow balls, they loose the game, as at least one snowball melted away.

Implemented Gameplay Features

The below two subsections shortly describe how the given *gameplay* mechanics of the given groups, *compulsory* and *optional*, have been designed and realized.

Compulsory

Compulsory gameplay mechanics are related to functionalities that let the game in it's most rudimentary form come to life and be playable.

3D Geometry is implemented using the *assimp library*. Models from *opengame-art*, namely the artist Kenny, are being used. There are several non-primitive models being read from file and rendered, e.g. trees, cliffs, lamps and walls.
The final game should look somewhat comic-like, hence the models are of the so-called "low poly" look. For consistency in style only models made by the aforementioned artist were used; except for primitive shapes.

Playable & Advanced Gameplay can be considered as fulfilled by the current state of the game, as there is a player character that can walk through the game world. Furthermore it is possible to jump, collect snowballs and shoot snowballs which have been collected. These are then collectible again.
When transitioning from one cube side to another, the gravity adjusts accordingly and player and camera parameters (position, rotation) are adjusted accordingly.

Version from:
22/06/2022

Min. 60 FPS and Framerate Independence are achieved at all times. The former is more easily achievable due to the low poly count of the low models used. It can also be checked during runtime via the HUD; s. below.
With calculating the *delta time* each frame and considering this in certain rendering steps, e.g. movement, the framerate independence is assured.

The Win/Loose condition in *Polar Adventures* are, as described above, realized with a timer. Should a player manage to collect and deliver all (6) snowballs to the collection point, they win the game, with a certain score. The score depends on how strongly the snowballs have melted, i.e. t is related to the speed of the player.
Should the player take too long, they loose. This time can be set in the *settings.ini* as the variable *maxGameTime* in seconds.

Intuitive Controls are, as very commonly used, moving with the *WASD*-keys, jumping with *SPACE* and adjusting the view of the camera with *MOUSE* movement. Additionally the player can throw collected snowballs with the *left mouse button*, whereas the *H*-key toggles the HUD. The second HUD can be toggled with the M-key.
Movement and camera adjustment have a continuous effect. In contrary the player can only jump, when on ground and only shoot snowballs, when there are some in the pocket. The HUD can always be toggled.

The Illumination model is using multiple light source. There is at least one directional light and multiple spot lights, to have light in all corners of the cubic world. The lights positions are defined in *Constants.h* and evaluated in the associated shader files.
All models used have associated material values to them and come with normal vectors.

Textures are provided for the player character and the skybox. Mipmapping and and trilinear filtering are enabled. Due to style consistencies we chose to use as little textured objects as possible and thus not having a too large contrast with the low poly models. Textures of models are loaded and assigned in *Model.h* and Mesh.h.

Moving objects so far consider the snowballs that the player can throw from the pocket. These will then fly, fall to the ground and roll around. Note that these also adjust their gravity, when transition to another cube side.
In the future there will be some moving platforms, for a platforming passage on the back side of the cube.

The Documentation is being presented with this document.

Adjustable parameters can be configured in the *settings.ini* file, e.g. toggling the debug view of the *bullet physics library.* This is done using the *inih library.*

Optional

Optional gameplay mechanics include any additional functionality, that enhances the experience or performance of the game, e.g. including a physics library or using view frustum culling.

Basic and Advanced Physics are simulated with the help of the bullet physics library. The player can not walk through any world object, e.g. trees or cliffs. Also thrown snowballs are colliding with all other solid objects. Furthermore dynamic objects, so far the snowballs and the player character controller, are included in the world. These move around based on the given velocity / WASD-keys pressed and adjust gravity w.r.t. to the cube side.
Collision callbacks are present in two forms. On the one hand, when the player and a snowball collide the latter is removed from the game world, i.e. collected and added to the players pocket. On a throw call (R-key) the snowball is added to the world with a certain throw speed and direction.
And on the other hand, if a snowball is colliding with the collection point, it is removed from the game world and added to the saved snowballs list.

The View-Frustum Culling The view frustum culling is implemented via an algorithm utilizing plane intersections, The main point here is, that a basic boundary box is found for each mesh via a min/max algorithm on each axis. That boundary box is then transformed, that is tested against certain planes of the frustum.

A HUD that can be toggled on and off at any time shows some game related information, such as the current snowballs in the world and collected by the player. Furthermore there is a new second HUD which shows how much the jump is charged and a map indicating on which side of the cube there are still snowballs left.

Implemented Effects

A CPU Particle System is implemented to create abstract white-ish meshes, that simulate snow. It is used to represent a *snow storm* on the back side. This snow will stop, when the five other snow balls have been collected. Then the 6th snowball on the bottom appears, and with the *snow storm* gone, the player can easily collect the last snowball and deliver it.

GPU Vertex Skinning from the animation effects group is used to display a walk animation on the player character object.

An Environment Map is used on the snowballs, which reflects the skybox (galaxy) and makes the snowballs look like little "galaxy balls".

Cel Shading from the shading effect group has been implemented. The models are rendered using a relatively simple cel-shading algorithm.

Edge Detection highlights the edges of the low-poly objects with a with outline, giving the world a more outstanding look.

Additional content

This section covers "free" content, i.e. effects, sounds etc. that enrich the game, but do not yield points.
So far there is some music integrated (background, jump, walk, collect, deliver and shoot sounds) and a skybox for a background image.

Tutorials and References

A list of tutorials and references that have been used can be found in the README of the projects github repository. The specific parts of the code that make use of a certain tutorial or pages of a book are then marked with comments in the source code.

Dependencies

In this section, the method of resolving dependencies is described and additionally for each external library their use case is outlined.
VCPKG is used to resolve any dependencies. The solution is built for 64-bit systems. The shell file *add_dependencies.bat* can be called to resolve all dependencies for the project. It assumes the OS to be windows and then installs all x64 versions of the dependencies.

GLEW and GLFW are (as usually) included to easily resolve general openGl calls.

GLM is used for mathematical operations.

Assimp is a library for loading model files. The necessary information is saved into specific data structures (*Model.h*, *Mesh.h*).

STB is utilized to load texture files and use them in models loaded with the above mentioned *assimp* library.

Bullet3 is a sophisticated physics library that offers an enormous pool of functionality. So far the project uses it's functionalities for collision detection, gravity simulation, collision callbacks and a kinematic character controller. The latter manages movement, jumping and several checks of the player object, such as being on ground or in mid air.

IrrKlang is a closed source library, that is used to play music in the game.

Inih is used to load the settings.ini file.

Freetype is used for loading fonts (.ttf files) and render these to display the HUD.

Note that in order to make the bullet3 library, installed via vcpkg, fully work, it is necessary to set the path pointing to the bullet directory in vcpkg to the additional include libraries in VS, in our case e.g. $C\backslash Dev\backslash vcpkg\backslash installed\backslash x64-windows\backslash include\backslash bullet$. Furthermore the libraries assimp and irrKlang needed to be included manually, i.e. putting source in the include directory, adding the libs and the dlls.

Hints, Tips and Cheats

The collection point is in the middle of the top side of the cube, depicted as a snowman, between two trees.

Snowball locations are, w.r.t the top side of the cube (the side the player is on, when starting the game): in the right corner in a crater, on the back side of the cube on a platform, on the right side of the cube in a "labyrinth" , on the front side of the cube on the highest palm tree, on the left side of the cube inside the most centered stone block (you can walk inside from one side) and on the bottom side the snowball appears on the plateau, when all other snowballs have been collected.