Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Computer Programming

«Pool 2D»

author	Michał Ferenc
instructor	dr inż. Piotr Fabian
year	2020/2021
lab group	section 2
deadline	27.06.2021

1. Task

Assignment 109

Pool in 2D. Write a program simulating the movement of balls in a 2D billiard game. Results should be presented as an animation (use the Allegro library). Details to discuss.

2. Analysis of the problem

2.1 Data structures:

Most of the data, like initial locations of balls is hard-coded in constant size arrays. Pointers to balls and their textures are stored in dynamic vectors for the purpose of ease of iterating through them using range based for loops.

2.2 Physics of balls:

For calculation of position balls are represented as points moving through 2D space with some velocity and constant deceleration due to friction. For checking of collisions each ball is assumed to have the same diameter of 28 pixels and equal masses (then they can be omitted when calculating velocities after the collision). Each collision is assumed to be perfectly elastic (kinetic energy and momentum are conserved). For the simulation to be run in real time the temporal resolution is variable and dependent on the time that it took to render previous frame. This approach requires small corrections of positions of balls during, resulting from too high time difference, to be resolved locally, by calculating exact time of collision, position of balls at the collision and change in velocities for this exact moment and then positions of balls at the initial time of calculation, effectively going back and forth in time. Another approach basing on finding and aggregating those critical points in time and then calculating state of the system in each one of them, in addition to regular time steps, could result in speed ramp effect (speed of the animation going up and down) when displayed at constant refresh rate. Second approach would provide more accurate simulation, but it would also require post processing compensating high differences in time differences between frames, which makes it unfeasible to use in real time game.

2.3 Pool rules:

There are 16 balls - cue ball (white), eighth ball (black), seven solid colored balls and seven striped ones. Initially all of the balls except cue ball are set up in triangle on the table. There are two players involved. Player 1. begins with "break shot" scattering the balls on the table. First ball pocketed after the break shot decides who plays with solids and who plays with stripes. On the beginning of each turn player sets the direction in which he wants to move the ball by moving the cue, then he decides how hard he wants to strike it using power bar on the right. If player succeeds to pocket his ball he gets another turn unless he also commits a foul. Foul is committed when: player fails to strike his own ball first, no ball hits the cushion, cue ball doesn't hit any other ball or cue ball is pocketed. Committing foul allows the opponent to move the cue ball in the beginning of his next turn.

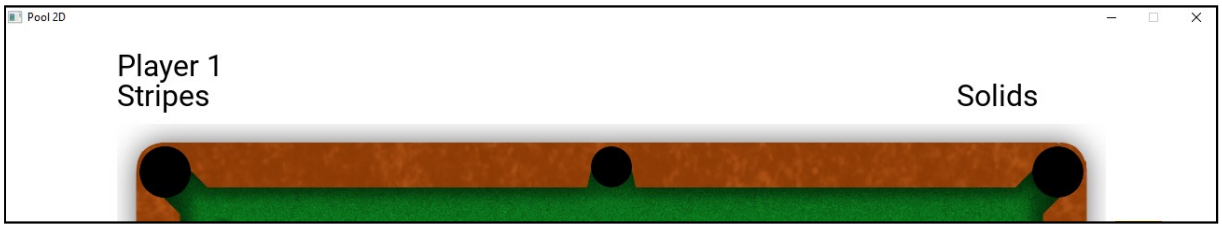
Pocketing black ball after "break shot" ends the game. If player pocketing it has previously pocketed all of his balls and doesn't commit a foul he wins.

3. External specification

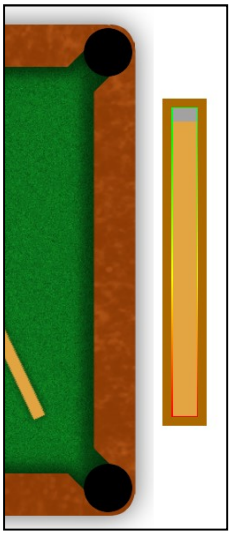
3.1 Running the program:

This is GUI program working in MS Windows operating system. For the program to function correctly *.dll libraries provided in github repository should be located in the same directory as executable file, or in location included in PATH environmental variable and the graphics should be located in gfx subdirectory. The game can be either launched through Windows Explorer or through the command line, when user wants more insight when the program doesn't work properly. In case of some known failure during startup of the program appropriate message is displayed on console output.

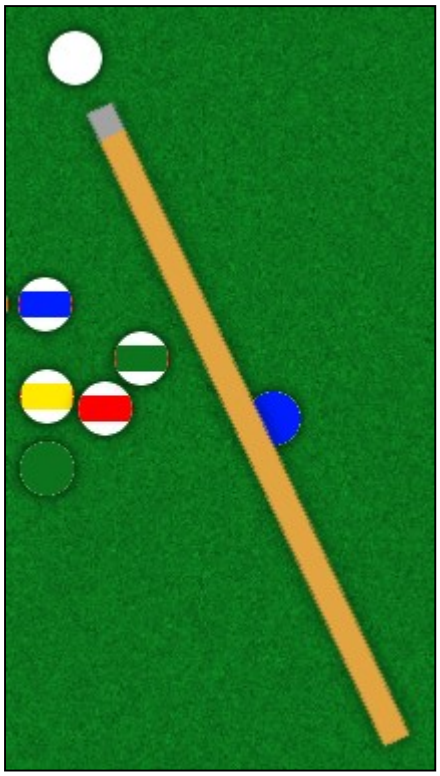
3.2 Game interface:



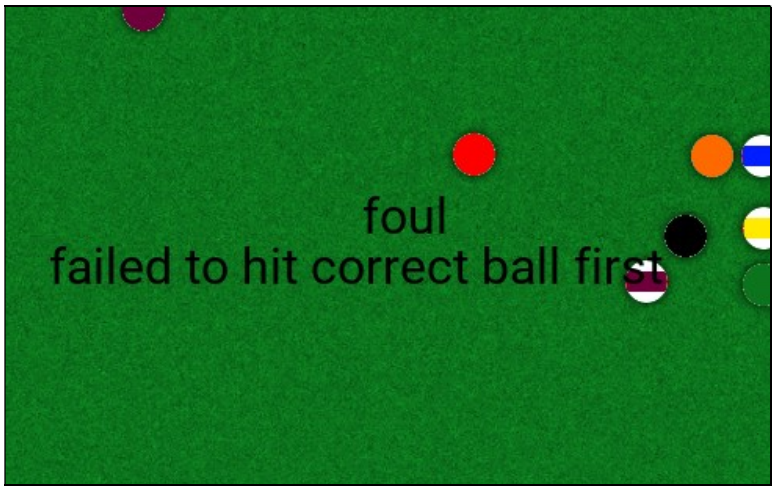
Above the table there is indicator whose turn it is now, along with their ball type if assigned



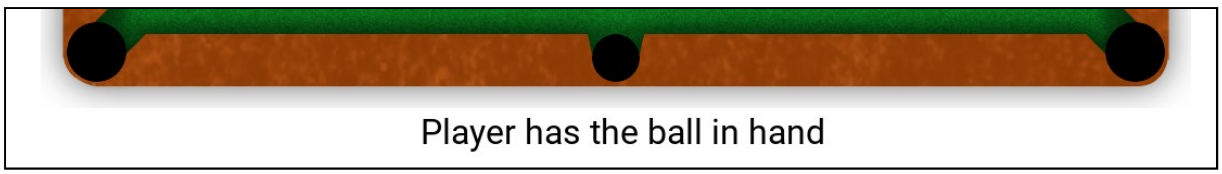
On the right of the table there is a powerbar - it is used to set how hard cue ball will be striked, it can be adjusted either by dragging it or up/down arrows



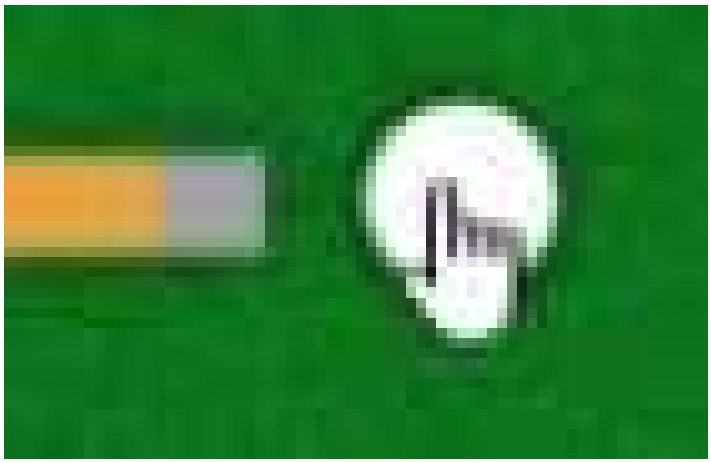
Cue and cue ball - position of cue relative to the ball determines the direction where cue ball will go when hit by cue. The cue can be either dragged around white ball using mouse or moved using left/right arrows. To launch the ball after setting the direction and force Player should press spacebar



When a foul is committed, there is a message displayed on the middle of the screen along with the justification.



When one of the players commits a foul the other player has the ball in hand, which means he can move it.



Each object that can be moved using mouse causes mouse pointer hovering over it to change shape

Once the game is over player indicators are replaced by winner indicator. Program stops reacting to any input besides window controls and needs to be restarted for a new game.

4. Internal specification

The program is implemented with OOP paradigm. WinMain function creates new object of class Game and calls it's method run, containing main game loop.

Game

Game- main class of the program

constructor - initializes SDL, loads textures, creates balls, bands and text messages;
run - here is placed main loop of the program. In each iteration SDL events are processed by Game::handle_events function, then game elements are rendered by function Game::render and time elapsed from previous iteration is calculated, it is then used to hide timeouted text fields and to perform physics calculations. After updating positions and speeds of the balls there are performed checks if: ball has been pocketed or all of the balls have stopped, then the actions are taken according to the rules of pool. Next there are performed checks for collisions between balls and balls with bands and correct changes of speeds applied. Collisions with bands and first ball hit by cue ball are remembered, as it is important for determining if the foul has or has not been committed. After this, if black ball has been pocketed program decides who has won the game, after that game goes into "END" state and reacts only to window controlls. Otherwise the program decides whose turn is next based on information if current player has pocketed at least one of his balls and if he committed a foul.
destructor - takes care of destroying textures and proper release of SDL resources for the program to finish gracefully
state variable of enumerative type: WAITING - waiting for players interaction, MOVING_BALL - player moves the ball right now , MOVING_CUE - player moves the ball right now, SETTING_POWER - player

sets the power of the strike, ROLLING - balls are in move, all interactions are checked, END - the game is over, program doesn't react to any interaction beside window controls.
There are also variables used to track if the rules of Poll are being followed, who should play in given moment, with which balls and who wins the game. They are described in the appendix.

Segment

Class Segment represents mathematical segment and is used to describe boundaries of the pool table.
Constructor of class Segment takes as arguments coordinates of initial point and terminal point, they are then converted to length of the segment, initial point and direction vector of the segment.
Segment::check_collision indicates if the ball has collided with given boundary by checking if parts of the vector between initial point of segment and position of given ball parallel to direction and normal vector of segment are less than: length of segment and radius of ball.

TextField

Class TextField is responsible for rendering text on screen.
Constructor takes as arguments: pointer to rendering context, text to be rendered, pointer to font that shall be used, coordinates of top left corner. Using this data the texture containing given text written in given font is generated. Rest of information is stored for future use. Text field is by default hidden and doesn't time out.
TextField::setTimeout is used to set value of timeout in milliseconds, -1 means that text field will never time out
TextField::hide and TextField::show are responsible for changing visibility of the text field
function draw renders displays text on screen and decreases time on screen by dt given in argument

Ball

Class Ball represents basic billiard ball, stores its id(balls in pool are numbered), position and velocity as well as flags describing state of the ball: ball is on table, ball is moving, ball has hit band in this turn.
constructor just initializes values
Ball::update changes position and velocity according to time difference dt given in argument
Ball::draw renders ball in correct place, pointer r is used to obtain rendering context
Ball::collide apply position and velocity changes resulting from collisions with bands and other balls
Ball::check_for_pocket checks if the ball has fallen into any of the pockets and if so, invokes method reacting to this event
Ball::pocket removes ball form the table

CueBall

class CueBall is derived from class Ball
constructor call base class constructor with coordinates and id 0 and initializes flag movable to be false
flag movable is set when player can move the ball
CueBall::move is used to change balls position if it is movable
CueBall::strike forces change of balls velocity

Other classes

classes EightBall, Solid and Stripe are classes derived from Ball mainly for the purpose of identifying them during game using RTTI. EightBall has also modified constructor, because it's id is always 8

5. Testing

The program has been tested by playing the game many times trying to perform legal and illegal moves. Program mostly behaves as it should, but during the testing it came out that it is impossible to win the game following rules. In last stage of game when player has pocketed all of his balls player should be able to hit black ball first, as there are none of his balls left, but the game doesn't allow it. It is caused by the fact, that when player hits ball other than his the game recognizes it as foul and checking if it was eight ball is omitted. Program was also tested with respect to low temporal resolution of simulation. Physics have proven to be unstable when subjected to artificially high time step, but tests on real hardware - both with integrated and dedicated GPU's have shown, that time differences between frames aren't nearly as high and the game runs properly. The game has also been tested with respect to running alongside stress test pushing the usage of the CPU to maximum and no difference in game's behavior was observed.

6. Conclusions

Creating this project has shown that creating fast, reliable and accurate real time physics simulation while keeping track on properties related to Eight Ball Pool is a challenging task requiring knowledge of target platform, it's strengths and downsides and needs to be tested by being subjected to sometimes unrealistic edge-case scenarios.

Appendix
Description of types and functions