COMP 2012H: Honors Object-Oriented Programming and Data Structures
Programming Assignment 3: Pipe Game
Source Code Documentation

Additional variables and functions

Self-defined variables and functions are only added in gameinstance.h(.cpp).

Public variables

```
enum Direction {UP, RIGHT, DOWN, LEFT};
enum PipeResult {SUCCESS, LEAKAGE, FAIL};
struct Coordinate { int y; int x; };
```

Private variables

```
QTimer* timer;
std::queue<Coordinate>* q;
int waterpressure[MAP_SIZE][MAP_SIZE];
bool isGameOvered;
```

Private functions

```
int getOutputDirection(int y, int x, Block* blocks[MAP_SIZE][MAP_SIZE], Direction dir[4]);
```

Return value: number of output direction
Parameters:

int y	Y-coordinate of pipe
int x	X-coordinate of pipe
Block* blocks[][]	Array of pipe
Direction dir	The output directions of corresponding pipe

```
bool isLeakageExist(Direction from, int y, int x, Block* blocks[MAP_SIZE][MAP_SIZE]);
```

Return value: any leakage from this pipe
Parameters:

Direction from	The direction where the water from
int y	Y-coordinate of pipe
int x	X-coordinate of pipe
Block* blocks[][]	Array of pipe

```
PipeResult piping(Direction from, int y, int x, Block* blocks[MAP_SIZE][MAP_SIZE], int val,
int pressure[MAP_SIZE][MAP_SIZE]);
```

Description: go through all the pipes until getting a result.
Return value: the result of game (success, leakage, fail)
Parameters:

Direction from	The direction where the water from
int y	Y-coordinate of pipe
int x	X-coordinate of pipe
Block* blocks[][]	Array of pipe
int val	The water pressure of corresponding pipe
int pressure[][]	Array of water pressure.

```
bool isToward(Direction dir, Direction dirs[], int numOutput);
bool isToward(int y, int x, Direction Dir);
```

Return value: whether the pipe can output the water to Direction dir.
Parameters:

Direction dir	The target direction
int y	Y-coordinate of pipe
int x	X-coordinate of pipe
Direction dir[]	Array of output directions of pipe
int numOutput	The number of output of corresponding pipe

```
bool isConnected(int y, int x, Direction dir);
```

Return value: whether the pipe is connected to the neighbour pipe.
Parameters:

int y	Y-coordinate of pipe
int x	X-coordinate of pipe
Direction dir	Is connected to this direction

```
void enqueue(int y, int x, std::queue<Coordinate>& q, int pressure[MAP_SIZE][MAP_SIZE]);
void dequeue(std::queue<Coordinate>& q, int pressure[MAP_SIZE][MAP_SIZE]);
```

Description: helper functions for animation
Parameters:

int y	Y-coordinate of pipe
int x	X-coordinate of pipe
queue<Coordinate> q	The queue storing the coordinate of blocks that to be process
int pressure[][]	The water pressure of pipe

```
void displayResult();
```

Description: caller function of piping()

Part 2: Implementations

Parsing the map: `load_map()`

1. Skip the first `(dest_level - 1)` of map, read line until getting the `(dest_level - 1)th ']'`
2. Skip the first line of data `'['`
3. Read the following `MAP_SIZE` line of data, line by line
4. Ignore all the parenthesis `'(', ')',` and spaces `' '` from the data, using regular expression `"\)?,?\s*\(?"`
5. Initiate the block by pair of data.

Determine game result: `piping()`

1. Safe guard: the coordinates are out of bounds, `FAIL` if YES
2. Safe guard: the water is flowing from lower pressure to higher pressure, `FAIL` is YES
3. Get the output directions of the current pipe.
4. Check if the current pipe is leaking, terminate the whole process and `LEAKAGE` if YES.
5. Check if the current pipe is the last pipe and output to right, `SUCCESS` if YES.
6. Save `val` as the pressure of current pipe.
7. Recursive call for all output directions, terminate immediately if there is `LEAKAGE`
8. Return `FAIL` after finish all recursive call.

Animation: `updateBlockImage()`

1. Trigger the `QTimer` when done button is clicked.
2. `QTimer` will emit a event to execute `updateBlockImage()`
3. In `updateBlockImage(), dequeue()` the first block from queue, and enqueue the output blocks to queue.
4. Change the image of current pipe from empty to fill.
5. Check if game end according to:
    1. If the two neighbours of last pipe is not empty pipe, they are highlighted
    2. The last pipe is highlighted
6. Trigger the `QTimer` again for next round of animation
7. `displayResult()` if the game is ended (fulfilling the requirement in 5)