Technical Documentation

Table of Contents

- List of Components
- Game Board Details
  - Game Board Components
  - Example
  - Special Values
  - Create the Game Board
  - Miscelaneous Checks
  - Find Source and Destination Pipes
  - Visited Map Array
  - Valid Pipes Table
  - Check if Pipe is Valid
  - Find Objects on Game Board
  - Find Specific Kinds of Objects on Game Board
  - Figure Out Which Object the Player is Hovering Over
  - Highlight Space Under Player Cursor
  - Get Row/Col from Mouse Position
- Pipes
  - Check if Player Picked an Object from the Bar
  - Make Pipe Snap to Grid
  - Add Pipe to Game Board
  - Removing Pipes
- Money
  - Show the Money Bar
  - Assigning Money Per Level
  - Spending Money
  - Refunding Money

List of Components

- Game Board
- Pipes
- Fences
- Enemies

Game Board

- 11 rows x 20 columns
- Each grid block is 96px x 96px
- Total size: 1920px x 1056px

Components

Name	Type
grid_space	ds_map

Name	Type
game_board	2d array of grid_squares

```
grid_space {
    row: int
    col: int
    x_coord: int
    y_coord: int
    resident_object: instance of an object
    valid: boolean
    removable: boolean
}

game_board[int][int] = grid_space;
```

Variable	Use
row	The space's row position in the game_board array
col	The space's column position in the game_board array
x_coord	The x value of the top left corner of the space relative to the room's 0,0
y_coord	The y value of the top left corner of the space relative to the room's 0,0
resident_object	A reference to the instance of the object that is placed on this space
valid	Whether the player can place an object on the space
removable	Whether the player can remove the object from the space

Example

```
var block_size = 96;
var row = 0;
var col = 9;
var pipe_instance = instance_create(x, y, obj_pipe);

var space = ds_map_create();
space[? "row"] = row;
space[? "col"] = col;
space[? "x_coord"] = col * block_size;
space[? "y_coord"] = row * block_size;
space[? "resident_object"] = pipe_instance;
space[? "valid"] = false;
space[? "removable"] = true;

game_board[row][col] = space;
```

This example creates one space and assigns it to the 10th column (0 is first) in the first row on the game board. It is marked as invalid, meaning the player cannot place a pipe there (because there already is a pipe there). However, it is marked removable, so that pipe could be deleted and replaced.

Special Values

- Source/Destination Pipes
  - Each level has two pipes already placed. These are the pipes that the pipeline needs to connect.

- These will be marked in the game_board array by spaces containing instances of the pipe_source and pipe_destination objects, respectively.

Create the game board

```
//In create event
var row;
var col;
var space;
for(row = 0; row < max_rows; row++){
    for(col = 0; col < max_cols; col++){
        space = ds_map_create();
        space[? "row"] = row;
        space[? "col"] = col;
        space[? "x_coord"] = col * block_size;
        space[? "y_coord"] = row * block_size;
        space[? "resident_object"] = noone;
        space[? "valid"] = true;
        space[? "removable"] = false;

        game_board[row, col] = space;
    }
}
```

Loop through game_board to do miscelaneous checks

```
var i;
var j;
for(i = 0; i < array_height_2d(game_board); i++){
    for(j = 0; j < array_length_2d(game_board, i); i++){
        //checks
    }
}
```

Find the source and destination pipes

```
//In create event for the room
//Find the source and destination pipes
source_struct = noone;
destination_struct = noone;

var source_found = false;
var destination_found = false;

var current_space;
var obj;
var obj_index;
var row;
var col;

for(row = 0; row < max_rows; row++){
    for(col = 0; col < max_cols; col++){
        if(source_found && destination_found){
            break;
        }

        current_space = game_board[row, col];
        obj = current_space[? "resident_object"];
        if(obj == noone){
            continue;
        }
        obj_index = obj.object_index;
        if(obj_index == obj_pipe_source){
            source_struct = current_space;
```

```
            source_found = true;
        }else if(obj_index == obj_pipe_destination){
            destination_struct = current_space;
            destination_found = true;
        }
    }
}

current_row = source_struct[? "row"];
current_col = source_struct[? "col"];
```

Create the visited_map 2d array

Create a 'visited_map' array to keep track of the places the algorithm has altready checked.

```
//In step event, right before the valid pipeline check
var row;
var col;
for(row = 0; row < max_rows; row++){
    for(col = 0; col < max_cols; col++){
        visited_map[row, col] = false;
    }
}
```

Valid Pipes Table

Next, depending on the pipe we're currently on, check around it to see if there is a valid pipe.

Pipe	Direction to check	optional valid pipe in that position
obj_pipe_up	up	obj_pipe_up, obj_pipe_corner_tr, obj_corner_tl
obj_pipe_up	down	obj_pipe_up, obj_pipe_corner_br, obj_pipe_corner_bl
obj_pipe_down	up	obj_pipe_up, obj_pipe_corner_tr, obj_pipe_corner_tl
obj_pipe_down	down	obj_pipe_up, obj_pipe_corner_br, obj_pipe_corner_bl
obj_pipe_side / source	left	obj_pipe_side, obj_pipe_corner_tl, obj_pipe_corner_bl
obj_pipe_side / source	right	obj_pipe_side, obj_pipe_corner_tr, obj_pipe_corner_br, obj_pipe_destination
obj_pipe_corner_tl	down	obj_pipe_up, obj_pipe_corner_br, obj_pipe_corner_bl
obj_pipe_corner_tl	right	obj_pipe_side, obj_pipe_corner_tr, obj_pipe_corner_br, obj_pipe_destination
obj_pipe_corner_tr	down	obj_pipe_up, obj_pipe_corner_br, obj_pipe_corner_bl
obj_pipe_corner_tr	left	obj_pipe_side, obj_pipe_corner_tl, obj_pipe_corner_bl
obj_pipe_corner_br	up	obj_pipe_up, obj_pipe_corner_tr, obj_pipe_corner_tl
obj_pipe_corner_br	left	obj_pipe_side, obj_pipe_corner_tl, obj_pipe_corner_bl
obj_pipe_corner_bl	up	obj_pipe_up, obj_pipe_corner_tr, obj_pipe_corner_tl

Pipe	Direction to check	optional valid pipe in that position
obj_pipe_corner_bl	right	obj_pipe_side, obj_pipe_corner_tr, obj_pipe_corner_br, obj_pipe_destination

This table also works since some pipes share valid pipes in specific directions. The algorithm below is based on this second table.

pipe	direction	valid pipes
obj_pipe_up, obj_pipe_down, obj_pipe_corner_bl, obj_pipe_corner_br	up	obj_pipe_up, obj_pipe_corner_tr, obj_corner_tl
obj_pipe_up, obj_pipe_down, obj_pipe_corner_tl, obj_pipe_corner_tr	down	obj_pipe_up, obj_pipe_corner_br, obj_pipe_corner_bl
obj_pipe_side, obj_pipe_corner_tr, obj_pipe_corner_br	left	obj_pipe_side, obj_pipe_corner_tl, obj_pipe_corner_bl
obj_pipe_side, obj_pipe_corner_tl, obj_pipe_corner_bl	right	obj_pipe_side, obj_pipe_corner_tr, obj_pipe_corner_br

Check for a valid pipeline

How this works needs explanation:

This algorithm is tracing the pipeline from the source pipe to the destination pipe. It starts at the source and checks if there is a pipe connected to it. The source is a side pipe, so there are only three possible pipes that could connect to it.

If there is a valid pipe connected to it, the algorithm moves to that space and continues checking the other directions. Important: It only checks places it has not already visited, as marked in the visited_map array.

This process repeats until the algorithm either lands on the destination, or it reaches a pipe which does not have a valid pipe connected to it. For example: If an up pipe had a side pipe above it and the bottom was visited, the algorithm would exit, setting the invalid flag.

```
//In step event
for(row = 0; row < max_rows; row++){
    for(col = 0; col < max_cols; col++){
        visited_map[row, col] = false;
    }
}

current_row = source_struct[? "row"];
current_col = source_struct[? "col"];

var done = false;
var space;
var pipe;
var next_space;
var next_pipe;
var old_row;
var old_col;

while(!done){
    old_row = current_row;
    old_col = current_col;

    visited_map[current_row, current_col] = true;
```

```
space = game_board[current_row, current_col];
pipe = space[? "resident_object"];

if(pipe != noone && pipe.object_index == obj_pipe_destination){
    valid = true;
    done = true;
    break;
}

//Check right
if(current_col + 1 < max_cols && visited_map[current_row, current_col + 1] == false){

    next_space = game_board[current_row, current_col + 1];
    next_pipe = next_space[? "resident_object"];

    if(next_pipe != noone){
        if(pipe.object_index == obj_pipe_source
        || pipe.object_index == obj_pipe_side
        || pipe.object_index == obj_pipe_corner_tl
        || pipe.object_index == obj_pipe_corner_bl){

            if(next_pipe.object_index == obj_pipe_side
                || next_pipe.object_index == obj_pipe_corner_tr
                || next_pipe.object_index == obj_pipe_corner_br
                || next_pipe.object_index == obj_pipe_destination){

                current_col += 1;
            }
        }
    }
}

//Check left
if(current_col - 1 >= 0 && visited_map[current_row, current_col - 1] == false){

    next_space = game_board[current_row, current_col - 1];
    next_pipe = next_space[? "resident_object"];

    if(next_pipe != noone){
        if(pipe.object_index == obj_pipe_side
        || pipe.object_index == obj_pipe_corner_tr
        || pipe.object_index == obj_pipe_corner_br){

            if(next_pipe.object_index == obj_pipe_side
                || next_pipe.object_index == obj_pipe_corner_tl
                || next_pipe.object_index == obj_pipe_corner_bl){

                current_col -= 1;
            }
        }
    }
}

//Check up
if(current_row - 1 >= 0 && visited_map[current_row - 1, current_col] == false){

    next_space = game_board[current_row - 1, current_col];
    next_pipe = next_space[? "resident_object"];

    if(next_pipe != noone){

        if(pipe.object_index == obj_pipe_up
            || pipe.object_index == obj_pipe_down
            || pipe.object_index == obj_pipe_corner_br
            || pipe.object_index == obj_pipe_corner_bl){

            if(next_pipe.object_index == obj_pipe_up
                || next_pipe.object_index == obj_pipe_down
                || next_pipe.object_index == obj_pipe_corner_tr
                || next_pipe.object_index == obj_pipe_corner_tl){

                current_row -= 1;
            }
        }
```

```
        }
    }

    //Check down
    if(current_row + 1 < max_cols && visited_map[current_row + 1, current_col] == false){

        next_space = game_board[current_row + 1, current_col];
        next_pipe = next_space[? "resident_object"];

        if(next_pipe != noone){

            if(pipe.object_index == obj_pipe_up
                || pipe.object_index == obj_pipe_down
                || pipe.object_index == obj_pipe_corner_tr
                || pipe.object_index == obj_pipe_corner_tl){

                if(next_pipe.object_index == obj_pipe_up
                    || next_pipe.object_index == obj_pipe_down
                    || next_pipe.object_index == obj_pipe_corner_br
                    || next_pipe.object_index == obj_pipe_corner_bl){

                    current_row += 1;
                }
            }
        }
    }

    //If all the directions have been checked and we haven't moved, invalid pipe
    if(old_row == current_row && old_col == current_col){
        done = true;
        valid = false;
        break;
    }
}
```

Find Objects on Game Board

This code constantly goes through each space on the game board and checks if there is an object there. If there is, it updates that space's `resident_object` value to match. It essentially keeps the data up-to-date with what's happening on screen.

```
//In step event
var instance;
for(row = 0; row < max_rows; row++){
    for(col = 0; col < max_cols; col++){
        space = game_board[row, col];
        instance = instance_position(space[? "x_coord"] + floor(block_size / 2),
                                     space[? "y_coord"] + floor(block_size / 2),
                                     all);

        //Do not overwrite any existing objects
        if(space[? "resident_object"] == noone){
            space[? "resident_object"] = instance;
        }

        if(instance != noone){
            space[? "valid"] = false;
        }
    }
}
```

Search for only select objects

In the `instance_position()` call the keyword `all` is used. As it is now, this script will find every object in the room. To only find select objects, create an empty parent object (do not need to create an instance at runtime) set this object as the `parent` to all the ones to search for.

To search for all the pipes on the game board, do the following:

1. Create an empty object called `pipe_parent` in GameMaker's object folder.
2. For every pipe object, set its `parent` to be `pipe_parent`
3. In the `instance_position()` function, replace `all` with `pipe_parent`

Replace `pipe_parent` with any other object (`water_parent` , `enemy_parent` , `fence_parent` , etc) to search for other objects.

Figure out which object the player is hovering over

```
//In create event
hover_instance = noone; //allows for global access

//In step event
hover_instance = instance_position(mouse_x, mouse_y, all);
```

The same trick for searching for only certain objects can be used here to only check for hover on certain objects.

```
pipe_hover_instance = instance_position(mouse_x, mouse_y, pipe_parent);
box_hover_instance = instance_position(mouse_x, mouse_y, box_parent);
```

Highlight the space under the player's cursor

Make the highlight object's depth a high value so it gets rendered first, thus, below other objects.

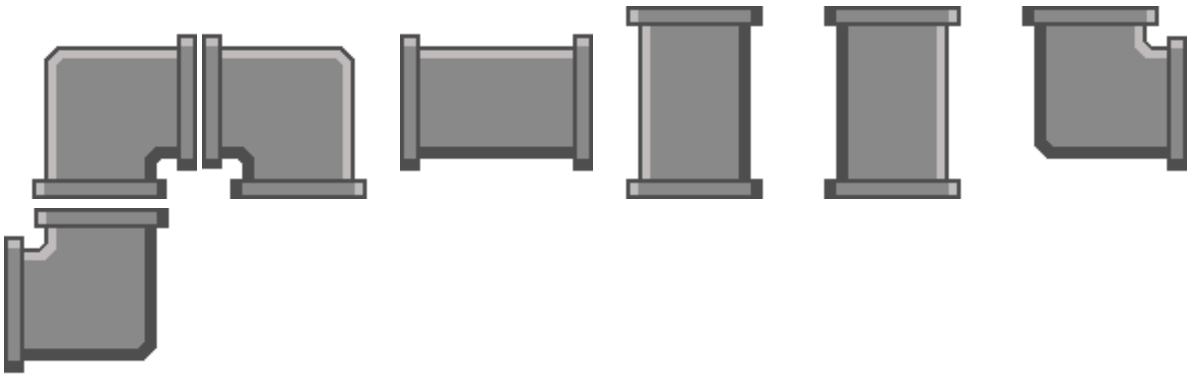
```
//In create event
highlight_space = instance_create(mouse_x, mouse_y, obj_highlight_space);

//In step event
highlight_space.x = floor(mouse_x/block_size) * block_size;
highlight_space.y = floor(mouse_y/block_size) * block_size;
```

Get row/col from mouse position to index the game_board

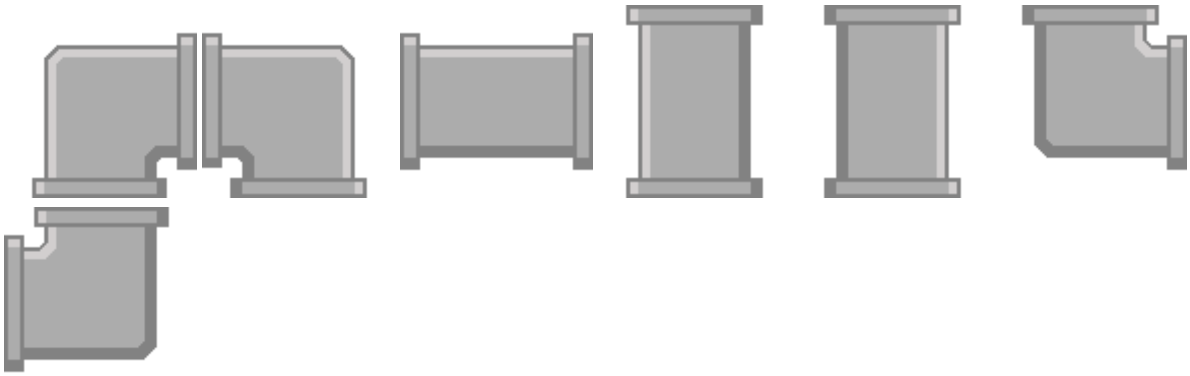
```
row = floor(mouse_y/block_size);
col = floor(mouse_x/block_size);
```

Pipes

There are 7 solid pipe types and 7 temporary ones. See below for the purpose of temporary pipes.

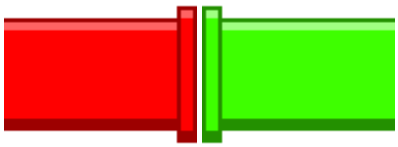


Each of those pipes as a temporary variant that are slightly more transparent.



There are two different pipes for up and down, simply to make the shadows line up in-game. They both work as each-other if the player does not care about such things. Placing an up piece instead of a down piece will have no negative consequences.

There are two special pipes which are not interactable: Source (red) and destination (green).



Check if player picked a pipe from the bar

In the step event there is a script that keeps track of which object the player is hovering over. The box_hover_instance variable describes which box along the bottom of the screen the player is hovering over. So, if they are hovering over an option, handle the input. Otherwise, do not do anything.

If the player is already holding a pipe, replace it with the one they just clicked.

```
if(mouse_check_button_pressed(mb_left)){
    if(box_hover_instance != noone){

        if(temp_pipe != noone){
            instance_destroy(temp_pipe);
            temp_pipe = noone;
        }

        var obj_to_make = noone;
        switch(box_hover_instance.object_index){
            case obj_box_pipe_up:
```

```
                obj_to_make = obj_pipe_up_temp;
                break;
            case obj_box_pipe_down:
                obj_to_make = obj_pipe_down_temp;
                break;
            case obj_box_pipe_side:
                obj_to_make = obj_pipe_side_temp;
                break;
            case obj_box_pipe_corner_tl:
                obj_to_make = obj_pipe_corner_tl_temp;
                break;
            case obj_box_pipe_corner_tr:
                obj_to_make = obj_pipe_corner_tr_temp;
                break;
            case obj_box_pipe_corner_br:
                obj_to_make = obj_pipe_corner_br_temp;
                break;
            case obj_box_pipe_corner_bl:
                obj_to_make = obj_pipe_corner_bl_temp;
                break;
            case obj_box_pipe_delete:
                obj_to_make = obj_box_pipe_delete_temp;
                break;
            default:
                break;
        }

        //If, for some strange reason, they picked an option that is not listed here, do nothing.
        if(obj_to_make != noone){
            temp_pipe = instance_create(mouse_x, mouse_y, obj_to_make);
        }

    }
}
```

Make Pipe Snap to Grid

When the player is dragging a new pipe onto the board it needs to snap to the grid. To do this, do the following:

```
if(temp_pipe != noone){
    temp_pipe.x = floor(mouse_x/block_size) * block_size;
    temp_pipe.y = floor(mouse_y/block_size) * block_size;
}
```

This will cause the pipe's x,y coordinates to move to the top left corner of the grid space. This only works if the pipe sprite's 0,0 is on the top left. Otherwise, some calculations would be needed to center the object in the grid square.

Add a new pipe to the board

Throughout gameplay the player will select a pipe from the bar at the bottom of the screen. They will be able to place an instance of that pipe wherever they click, provided the space is valid.

When the player clicks on the icon, create a temporary pipe at the mouse's location and snap it to the grid. Make this temporary piece follow the mouse's movements until the player clicks on the space where they want it to rest. The player does not have to hold down the mouse button to drag the piece. It is attached to the cursor when the player clicks the left mouse button and is released from the cursor when they click the right mouse button. This is to avoid cases of "Crap, I didn't mean to let go there!".

When the player clicks on a space, first check it to see if there is already an object there, or if the space is invalid. If either of these is true, the player cannot place the pipe there. If the desired space is empty, destroy the temporary pipe and place a solid pipe of the same type at that location. The solid pipe does not follow the mouse anymore and is now counted as part of the gameboard.

TL;DR: Temporary pipes are not part of the game board until the player clicks on a spot. Then they are replaced with a solid pipe of the same size, which is considered a part of the game board.

```
if(mouse_check_button_pressed(mb_right)){

    //First, check if it is a valid place
    var row = floor(mouse_y/block_size);
    var col = floor(mouse_x/block_size);
    if(row < max_rows && col < max_cols){
        var space = game_board[row, col];
        if(space[? "valid"] == true){
            if(temp_pipe != noone){

                var obj_to_make = noone;
                switch(temp_pipe.object_index){
                    case obj_pipe_up_temp:
                        obj_to_make = obj_pipe_up;
                        break;
                    case obj_pipe_down_temp:
                        obj_to_make = obj_pipe_down;
                        break;
                    case obj_pipe_side_temp:
                        obj_to_make = obj_pipe_side;
                        break;
                    case obj_pipe_corner_tl_temp:
                        obj_to_make = obj_pipe_corner_tl;
                        break;
                    case obj_pipe_corner_tr_temp:
                        obj_to_make = obj_pipe_corner_tr;
                        break;
                    case obj_pipe_corner_br_temp:
                        obj_to_make = obj_pipe_corner_br;
                        break;
                    case obj_pipe_corner_bl_temp:
                        obj_to_make = obj_pipe_corner_bl;
                        break;
                    default:
                        break;
                }

                if(obj_to_make != noone){
                    var _x = floor(temp_pipe.x / block_size) * block_size;
                    var _y = floor(temp_pipe.y / block_size) * block_size;

                    instance_create(_x, _y, obj_to_make);
                    space[? "valid"] = false;
                    space[? "removable"] = true;
                    game_board[row, col] = space;
                }
            }
        }
    }
}
```

Removing pipes

On the bar there is a box with a big red X. The player can click this and touch any pipe that they previously placed to delete it.

```
if(mouse_check_button_pressed(mb_right)){
    if(temp_pipe != noone && temp_pipe.object_index == obj_box_pipe_delete_temp){
        if(pipe_hover_instance != noone){
            var row = floor(mouse_y/block_size);
            var col = floor(mouse_x/block_size);
            if(row < max_rows && col < max_cols){
                var space = game_board[row, col];
                if(space[? "removable"] == true){
                    var object = space[? "resident_object"];
                    instance_destroy(object);
```

```
                    space[? "resident_object"] = noone;
                    space[? "valid"] = true;
                    visited_map[row, col] = false;
                    game_board[row, col] = space;
                    pipe_hover_instance = noone;
                }
            }
        }
    }
}
```

Money

The player has a limited amount of money to spend and each pipe costs money to build. Straight pieces cost $50 and corner pieces cost $100. When the level starts the player is given a sum of money. When they place a pipe, the cost is removed from that sum. When the player removes a pipe they get a full refund.

To make money the player must place pipes. If they place a pipe in a valid position, such that it contributes to the pipeline, then an alarm will start. Whenever that alarm triggers a money symbol will appear on top of the pipe. If the player left-clicks on the money it will be added to their total money.

Showing Available Money

This displays the player's available money at the bottom left of the screen.

```
//In create event
instance_create(96, 960, obj_box_money);
```

This draws the player's money within the borders of the money box in a gold color.

```
//In obj_box_money's draw_gui event
var money_color = make_color_rgb(255,214,0);

draw_set_font(fnt_money);
draw_set_color(money_color);
draw_set_halign(fa_center);
draw_set_valign(fa_center);

var _x = x + sprite_width/2;
var _y = y + sprite_height/2;
var money_local;
with(obj_controller){
    money_local = money;
}
draw_text(_x, _y, "Budget: $" + string(money_local));
```

Assigning Money Per Level

Each level starts with a different amount of money. This value is set when the room's controller is loaded.

```
//In create event
var temp = 0;
switch(room){
    case rm_level_0:
        temp = 50;
        break;
    case rm_level_1:
        temp = 500;
        break;
```

```
    case rm_level_2:
        temp = 1100;
        break;
    default:
        break;
}

money = temp;
```

Spending Money

Update the add pipe code to include money. In every case, include a `money_to_lose` variable that keeps track of how much each pipe costs. When the pipe is placed, subtract `money_to_lose` from the total money value.

```
if(mouse_check_button_pressed(mb_right)){

    //First, check if it is a valid place
    var row = floor(mouse_y/block_size);
    var col = floor(mouse_x/block_size);
    if(row < max_rows && col < max_cols){
        var space = game_board[row, col];
        if(space[? "valid"] == true){
            if(temp_pipe != noone){

                var obj_to_make = noone;
                var money_to_lose = 0;
                switch(temp_pipe.object_index){
                    case obj_pipe_up_temp:
                        obj_to_make = obj_pipe_up;
                        money_to_lose = 50;
                        break;
                    case obj_pipe_down_temp:
                        obj_to_make = obj_pipe_down;
                        money_to_lose = 50;
                        break;
                    case obj_pipe_side_temp:
                        obj_to_make = obj_pipe_side;
                        money_to_lose = 50;
                        break;
                    case obj_pipe_corner_tl_temp:
                        obj_to_make = obj_pipe_corner_tl;
                        money_to_lose = 100;
                        break;
                    case obj_pipe_corner_tr_temp:
                        obj_to_make = obj_pipe_corner_tr;
                        money_to_lose = 100;
                        break;
                    case obj_pipe_corner_br_temp:
                        obj_to_make = obj_pipe_corner_br;
                        money_to_lose = 100;
                        break;
                    case obj_pipe_corner_bl_temp:
                        obj_to_make = obj_pipe_corner_bl;
                        money_to_lose = 100;
                        break;
                    default:
                        break;
                }

                if(obj_to_make != noone){
                    var _x = floor(temp_pipe.x / block_size) * block_size;
                    var _y = floor(temp_pipe.y / block_size) * block_size;

                    instance_create(_x, _y, obj_to_make);
                    space[? "valid"] = false;
                    space[? "removable"] = true;
                    game_board[row, col] = space;

                    money -= money_to_lose;
                }
```

```
            }
        }
    }
}
```

Refunding Money

Update the remove pipe code to refund the price of the pipe being removed

```
if(mouse_check_button_pressed(mb_right)){
    if(temp_pipe != noone && temp_pipe.object_index == obj_box_pipe_delete_temp){
        if(pipe_hover_instance != noone){
            var row = floor(mouse_y/block_size);
            var col = floor(mouse_x/block_size);
            if(row < max_rows && col < max_cols){
                var space = game_board[row, col];
                if(space[? "removable"] == true){

                    var object = space[? "resident_object"];

                    //Refund the price of the pipe
                    var refund = 0;
                    switch(object.object_index){
                        case obj_pipe_up:
                            refund = 50;
                            break;
                        case obj_pipe_down:
                            refund = 50;
                            break;
                        case obj_pipe_side:
                            refund = 50;
                            break;
                        case obj_pipe_corner_tl:
                            refund = 100;
                            break;
                        case obj_pipe_corner_tr:
                            refund = 100;
                            break;
                        case obj_pipe_corner_br:
                            refund = 100;
                            break;
                        case obj_pipe_corner_bl:
                            refund = 100;
                            break;
                        default:
                            break;
                    }
                    money += refund;

                    instance_destroy(object);
                    space[? "resident_object"] = noone;
                    space[? "valid"] = true;
                    visited_map[row, col] = false;
                    game_board[row, col] = space;
                    pipe_hover_instance = noone;
                }
            }
        }
    }
}
```