Digital System Design and Implementation

HW #3

姓名:黃鉦淳 學號:108303013

a. Block diagram

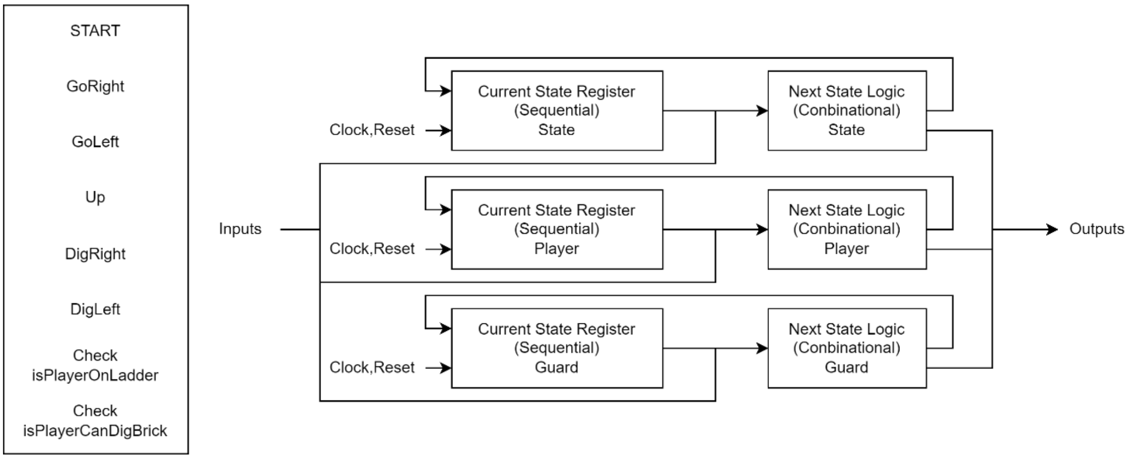
透過以下 2-Process FSM 完成此次作業。

Current State Register 主要是儲存 Next State Logic 具有記憶性的參數(ie. GuardX/GuardY/GuardDir/PlayerX/PlayerY/HoleX/HoleY/etc.)

Next State Logic 主要是負責該元素所有的狀態計算

Pseudo Code

```
// State
case (State)
    Stop:
        if (START) State_next = Movement;
    Movement:
        if (Guard_Next_State_Position_Trap_In_Hole) State_next = Trap;
        if (Guard_Next_State_Position_Hit_Player) State_next = Die;
        if (Player_Next_State_Position_Trap_In_Hole && HoleCount = 4)
            State_next = Die;
    Trap:
        if (TrapCount = 4) State_next = Movement;
    Die:
        if (Chance = 0) State_next = Stop;

// Guard
case (State)
    Movement:
        // Guard move the same Direction
        if (Guard_Next_State_Hit_Wall) GuardDir = GuardDir_invert;
        if (Guard_Next_State_Trap_In_Hole) TrapCount = TrapCount + 1;
    Trap:
        // Record GuardDir
        if (TrapCount = 4) Guard move the same Direction
    Die :
        // Initialize Guard Position

// Player
case (State)
    Movement, Trap:
    // Move the following instruction (Right/Left/Up)
    if (Player_Next_State_Get_Gold) Score = Score + 60;
    if (Player_Dig_Hole)
        // Record Hole Position, HoleCount = HoleCount +1;
```


START
GoRight
GoLeft
Up
DigRight
DigLeft
Check isPlayerOnLadder
Check isPlayerCanDigBrick
Inputs
Clock,Reset
Current State Register (Sequential) State
Next State Logic (Conbinational) State
Clock,Reset
Current State Register (Sequential) Player
Next State Logic (Conbinational) Player
Clock,Reset
Current State Register (Sequential) Guard
Next State Logic (Conbinational) Guard
Outputs

b. Verilog codes

```
`timescale 1ns / 1ps

module Main(CLK, RESET, START, GoRight, GoLeft, Up, DigLeft, DigRight,
                PlayerX, PlayerY, GuardX, GuardY, TrapCount, HoleCount, HoleX,
HoleY, Touch, Drop, Score, Chance);
input CLK, RESET;
input START, GoRight, GoLeft, Up, DigLeft, DigRight;
output reg [3:0] PlayerX, PlayerY;
output reg [3:0] GuardX, GuardY;
output reg [2:0] TrapCount, HoleCount;
output reg [3:0] HoleX, HoleY;
output reg Touch, Drop;
output reg [6:0] Score;
output reg [1:0] Chance;

reg [3:0] GuardX_next;
reg [3:0] HoleX_next, HoleY_next;

reg [3:0] GateX = 6, GateY = 6;
reg [3:0] GoldX = 9, GoldY = 4;
reg signed [1:0] PlayerXDir, PlayerYDir;                  // left=-1, still=0, right=1
reg signed [1:0] GuardXDir, GuardYDir;                      // left=-1, still=0, right=1
reg signed [1:0] GuardXDir_next, GuardYDir_next;     // left=-1, still=0, right=1
reg [2:0] HoleCount_add;
reg [6:0] Score_add;
reg [1:0] Chance_minus;
reg [2:0] TrapCount_add;
reg signed [1:0] GuardXDir_tmp, GuardXDir_tmp_next;
reg isPlayerDigHole;
reg isPlayerTrapInHole;
reg isGuardTrapInHole;
reg [3:0] PlayerX_next;
reg [1:0] State, State_next;
reg isGuardReset;
reg isHoleReset;

wire isPlayerOnLadder;
wire [1:0]isPlayerOverBrick;
Check_Ladder Check_Ladder(PlayerX, PlayerY, isPlayerOnLadder);
Check_Brick Check_Brick(PlayerX, PlayerY, isPlayerOverBrick);

parameter     State_Stop = 0,
                State_Movement = 1,
                State_Trap = 2,
                State_Die = 3;

parameter     GuardXDir_init = -1, // ID (odd) -> initial Guard move to the left
                GuardYDir_init = 0;
```

```
parameter     HoleX_init = 15,
                HoleY_init = 15;

// (a)
parameter     PlayerX_init = 7,
                PlayerY_init = 1,
                GuardX_init = 1,
                GuardY_init = 4;
// (b)
// parameter     PlayerX_init = 3,
//                  PlayerY_init = 1,
//                  GuardX_init = 2,
//                  GuardY_init = 1;

// Current State Register (sequential)
// State
always @(posedge CLK or posedge RESET)
begin
        if(RESET)
        begin
                State <= State_Stop;
                Chance <= 2;
        end
        else
        begin
                State <= State_next;
                Chance <= Chance - Chance_minus;
        end
end

// Guard
always @(posedge CLK or posedge RESET)
begin
        if(RESET)
        begin
                GuardX <= GuardX_init;
                GuardY <= GuardY_init;
                GuardXDir <= 0;
                GuardXDir_tmp <= 0;
                GuardYDir <= 0;
                TrapCount <= 0;
        end
        else
        begin
                if(isGuardReset)
                begin
                GuardX <= GuardX_init;
                GuardY <= GuardY_init;
                end
                else
                begin
                GuardX <= $signed(GuardX) + GuardXDir_next;
                GuardY <= $signed(GuardY) + GuardYDir_next;
                end

                GuardXDir <= GuardXDir_next;
                GuardXDir_tmp <= GuardXDir_tmp_next;
                GuardYDir <= GuardYDir_next;
                TrapCount <= TrapCount + TrapCount_add;
        end
end

// Player
always @(posedge CLK or posedge RESET)
begin
        if(RESET)
        begin
                PlayerX <= PlayerX_init;
                PlayerY <= PlayerY_init;
                HoleX <= HoleX_init;
                HoleY <= HoleY_init;
                Score <= 0;
                HoleCount <= 0;
        end
        else
```

```
        begin
                PlayerX <= $signed(PlayerX) + PlayerXDir;
                PlayerY <= $signed(PlayerY) + PlayerYDir;
                if(isHoleReset)
                begin
                        HoleX <= HoleX_init;
                        HoleY <= HoleY_init;
                end
                else
                begin
                        HoleX <= HoleX_next;
                        HoleY <= HoleY_next;
                end
                Score <= Score + Score_add;
                HoleCount <= HoleCount + HoleCount_add;
        end
end

// Next State Logic (combinational), Output Logic (combinational)
// State
always @(*)
begin
        case(State)
                State_Stop: begin
                        Touch = 0;
                        Chance_minus = 0;
                        if(START)
                                State_next = State_Movement;
                        else
                                State_next = State_Stop;
                end
                State_Movement: begin
                        Touch = 0;
                        State_next = State_Movement;
                        Chance_minus = 0;
                        GuardX_next = $signed(GuardX) + GuardXDir_next;

                        if(isPlayerDigHole && !isPlayerTrapInHole && GuardX_next ==
HoleX && GuardY == (HoleY + 1))
                    State_next = State_Trap;

               if(GuardX == PlayerX && GuardY == PlayerY)
               begin
                    State_next = State_Die;
                    Touch = 1;
                    Chance_minus = 1;
               end

               if(isPlayerTrapInHole && HoleCount == 4)
               begin
                    State_next = State_Die;
                    Touch = 1;
                    Chance_minus = 1;
               end
          end
          State_Trap: begin
               Touch = 0;
               Chance_minus = 0;

               if(TrapCount == 4)
                    State_next = State_Movement;
               else
                    State_next = State_Trap;
          end
          State_Die: begin
               Touch = 0;
               Chance_minus = 0;
               if(Chance == 0)
                    State_next = State_Stop;
               else
                    State_next = State_Movement;
          end
     endcase
end

// Guard
```

```
always @(*)
begin
        case (State)
                State_Stop: begin
                        Drop = 0;
                        TrapCount_add = 0;
                        isGuardTrapInHole = 0;
                        GuardXDir_next = 0;
                        GuardYDir_next = 0;
                        GuardXDir_tmp_next = 0;
                        isGuardReset = 0;
                end
                State_Movement: begin
                        Drop = 0;
                        TrapCount_add = 0;
                        isGuardTrapInHole = 0;
                        GuardXDir_next = GuardXDir;
                        GuardYDir_next = GuardYDir;
                        GuardXDir_tmp_next = 0;
                        isGuardReset = 0;

                        if(GuardX == 0)
                                GuardXDir_next = 1;
                        else if(GuardX == 9)
                                GuardXDir_next = -1;

                        if(!GuardXDir && !GuardYDir &&!isPlayerDigHole)
                        begin
                                GuardXDir_next = GuardXDir_init;
                                GuardYDir_next = GuardYDir_init;
                        end

                        GuardX_next = $signed(GuardX) + GuardXDir_next;
                        if(isPlayerDigHole && !isPlayerTrapInHole && GuardX_next ==
HoleX && GuardY == (HoleY + 1))
                        begin
                                GuardYDir_next = -1;
                                TrapCount_add = 1;
```

```
            end
        end
        State_Trap: begin
            Drop = (TrapCount == 1);
            TrapCount_add = (TrapCount == 4) ? ~TrapCount + 1 : 1; //
TrapCount = 0, overflow
            isGuardTrapInHole = 1;
            GuardXDir_tmp_next = (TrapCount == 1) ? GuardXDir :
GuardXDir_tmp;
            GuardXDir_next = (TrapCount == 4) ? GuardXDir_tmp : 0;
            GuardYDir_next = (TrapCount == 3);
            isGuardReset = 0;
        end
        State_Die: begin
            Drop = 0;
            TrapCount_add = 0;
            isGuardTrapInHole = 0;
            GuardXDir_tmp_next = 0;
            GuardXDir_next = 0;
            GuardYDir_next = 0;
            isGuardReset = (Chance != 0);
        end
    endcase
end

// Player
always @(*)
begin
    case (State)
        State_Stop: begin
            PlayerXDir = 0;
            PlayerYDir = 0;
            Score_add = 0;
            isPlayerDigHole = 0;
            isPlayerTrapInHole = 0;
            HoleCount_add = 0;
            HoleX_next = HoleX_init;
            HoleY_next = HoleY_init;
            isHoleReset = 1;
        end
        State_Movement, State_Trap: begin
            HoleX_next = HoleX;
            HoleY_next = HoleY;
            PlayerXDir = 0;
            PlayerYDir = 0;
            HoleCount_add = 0;
            Score_add = 0;
            isPlayerDigHole = 0;
            isPlayerTrapInHole = 0;
            isHoleReset = 0;

            if (PlayerX == HoleX && PlayerY == HoleY)
                isPlayerTrapInHole = 1;

            if(!isPlayerTrapInHole && GoRight)
                PlayerXDir = 1;

            if(!isPlayerTrapInHole && GoLeft)
                PlayerXDir = -1;

            if(Up && isPlayerOnLadder)
                PlayerYDir = 1;

            if(HoleX_next == HoleX_init && HoleY_next == HoleY_init)
                isPlayerDigHole = 0;
            else
            begin
                isPlayerDigHole = 1;
                HoleCount_add = 1;
            end

            PlayerX_next = PlayerX + PlayerXDir;
            if(isPlayerDigHole && !isGuardTrapInHole && PlayerX_next ==
HoleX && PlayerY == HoleY + 1)
                PlayerYDir = -1;
            if(PlayerX_next == GoldX && PlayerY == GoldY && PlayerXDir == 1)
                Score_add = 60;

            if(!isPlayerDigHole && DigLeft && isPlayerOverBrick[1])
            begin
                HoleX_next = PlayerX - 1;
                HoleY_next = PlayerY - 1;
                HoleCount_add = 1;
            end

            if(!isPlayerDigHole && DigRight && isPlayerOverBrick[0])
            begin
                HoleX_next = PlayerX + 1;
                HoleY_next = PlayerY - 1;
                HoleCount_add = 1;
            end

            if(isPlayerTrapInHole && HoleCount == 3)
                PlayerYDir = 1;

            if(TrapCount == 1)
                HoleCount_add = ~HoleCount + 1; // HoleCount = 0, overflow

            if(TrapCount > 1)
                HoleCount_add = 0;

            if(HoleCount == 4)
            begin
                PlayerYDir = 0;
                isPlayerDigHole = 0;
                HoleCount_add = ~HoleCount + 1; // HoleCount = 0, overflow
            end

            if(TrapCount == 4)
                isPlayerDigHole = 0;

            if(HoleCount == 4 || TrapCount == 4)
```

```
                isHoleReset = 1;
        end
        State_Die: begin
            PlayerXDir = 0;
            PlayerYDir = 0;
            HoleX_next = HoleX;
            HoleY_next = HoleY;
            HoleCount_add = 0;
            isPlayerDigHole = 0;
            isPlayerTrapInHole = 0;
            isHoleReset = 0;
            Score_add = 0;
        end
    endcase
end
endmodule

module Check_Ladder(PlayerX, PlayerY, isPlayerOnLadder);
input [3:0]PlayerX, PlayerY;
output reg isPlayerOnLadder;
always @(PlayerX or PlayerY)
begin
    if(PlayerX == 6 && 1 <= PlayerY && PlayerY <= 3)
        isPlayerOnLadder = 1;
    else if(PlayerX == 3 && 4 <= PlayerY && PlayerY <= 5)
        isPlayerOnLadder = 1;
    else
        isPlayerOnLadder = 0;
end
endmodule

module Check_Brick(PlayerX, PlayerY, isPlayerOverBrick);
input [3:0]PlayerX, PlayerY;
// isPlayerOverBrick[1] = 1 when Player can dig left Brick
// isPlayerOverBrick[0] = 1 when Player can dig right Brick
output reg [1:0]isPlayerOverBrick;

always @(PlayerX or PlayerY)
```

```
begin
        if(PlayerY == 1)
        begin
                if(PlayerX == 0 || PlayerX == 7)
                        isPlayerOverBrick = 2'b01;
                else if((1 <= PlayerX && PlayerX <= 4) || PlayerX == 6 || PlayerX == 8)
                        isPlayerOverBrick=2'b11;
                else if(PlayerX == 5 || PlayerX == 9)
                        isPlayerOverBrick = 2'b10;
                else
                        isPlayerOverBrick = 2'b00;
        end
        else if(PlayerY == 4)
        begin
                if(PlayerX == 0 || PlayerX == 4 || PlayerX == 7)
                        isPlayerOverBrick = 2'b01;
                else if(PlayerX == 1 || PlayerX == 3 || PlayerX == 6 || PlayerX == 8)
                        isPlayerOverBrick = 2'b11;
                else if(PlayerX == 2 || PlayerX == 5 || PlayerX == 9)
                        isPlayerOverBrick = 2'b10;
                else
                        isPlayerOverBrick = 2'b00;
        end
        else if(PlayerY == 6)
        begin
                if(PlayerX == 0 || PlayerX == 4)
                        isPlayerOverBrick = 2'b01;
                else if(PlayerX == 1 || PlayerX == 3 || (5 <= PlayerX && PlayerX <= 8))
                        isPlayerOverBrick = 2'b11;
                else if(PlayerX == 2 || PlayerX == 9)
                        isPlayerOverBrick = 2'b10;
                else
                        isPlayerOverBrick = 2'b00;
        end
        else
                isPlayerOverBrick = 2'b00;
end
endmodule
```

c. Test bench

```
`timescale 1ns / 1ps

module Main_tb();
reg CLK, RESET;
reg START, GoRight, GoLeft, Up, DigLeft, DigRight;
wire [3:0]PlayerX, PlayerY;
wire [3:0]GuardX, GuardY;
wire [2:0]TrapCount, HoleCount;
wire [3:0]HoleX, HoleY;
wire Touch, Drop;
wire [6:0]Score;
wire [1:0]Chance;

Main Main_tb(CLK, RESET, START, GoRight, GoLeft, Up, DigLeft, DigRight,
             PlayerX, PlayerY, GuardX, GuardY, TrapCount, HoleCount, HoleX,
HoleY, Touch, Drop, Score, Chance);

initial begin
    CLK = 1;
    START = 0;
    GoRight = 0; GoLeft = 0; Up = 0;
    DigRight = 0; DigLeft = 0;
    RESET = 0;
    #100; // Global Reset
    RESET = 1; #50; RESET = 0;
    #100; START = 1; #100; START=0;
    // (a) Player(x, y) = (7, 1) | Guard(x, y) = (1, 4)
    GoLeft = 1; #100; GoLeft = 0;
    Up = 1; #300; Up = 0;
    GoRight = 1; #300; GoRight = 0;
    DigLeft = 1; #100; DigLeft = 0;
    GoLeft = 1; #600; GoLeft = 0;
    #25; Up = 1; #175; Up = 0;
    GoRight = 1; #300; GoRight = 0;
    #50;
    // (b) Player(x, y) = (3, 1) | Guard(x, y) = (2, 1)
    // #200;
```

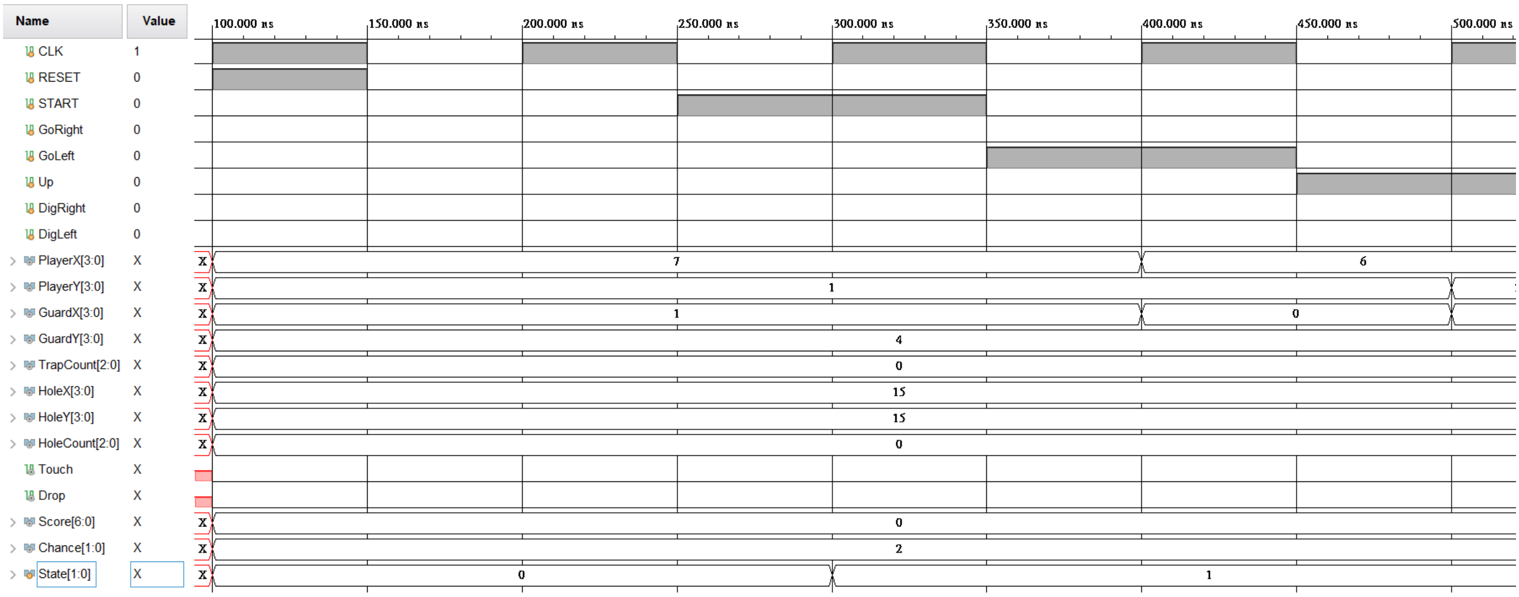
```
        // DigRight = 1; #100; DigRight = 0;
        // GoRight = 1; #100; GoRight = 0;
        // #1250;

        $finish;
end

always begin
        #50; CLK = ~CLK;
end
endmodule
```

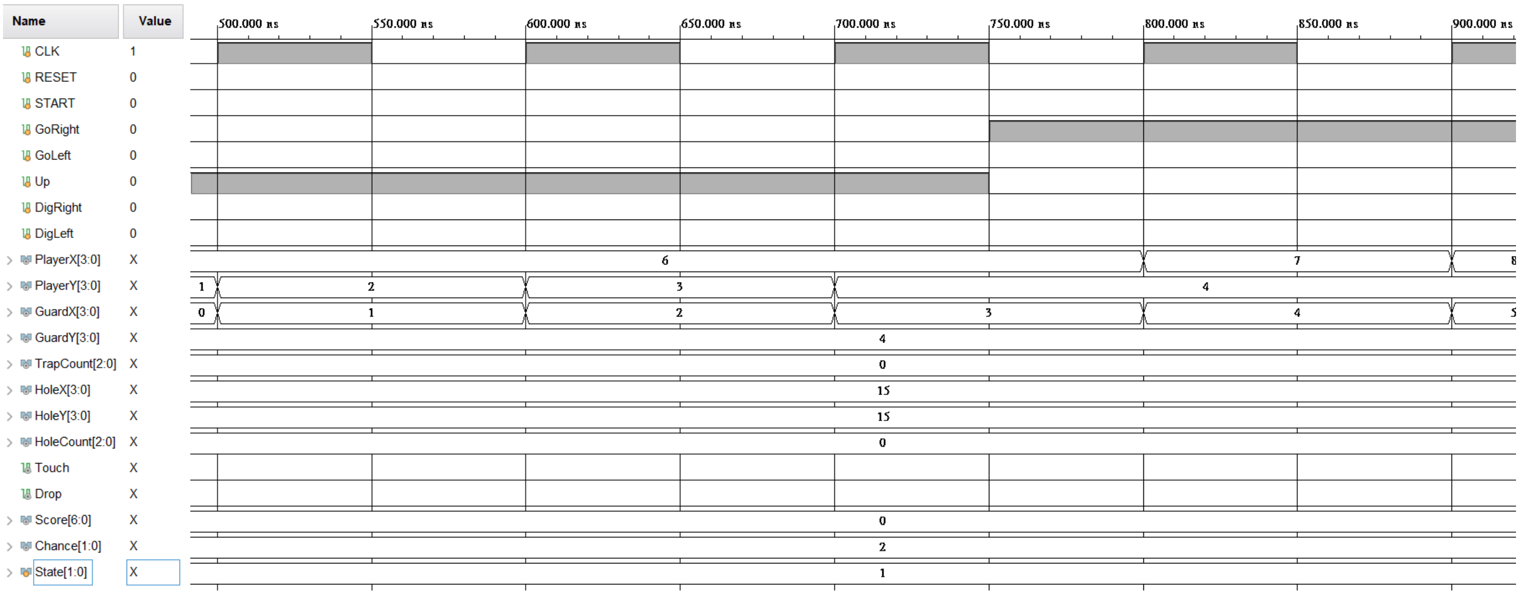
d. Input/output waveforms (behavior simulation and post-route simulation)

State => Stop = 0, Movement = 1, Trap = 2, Die = 3;

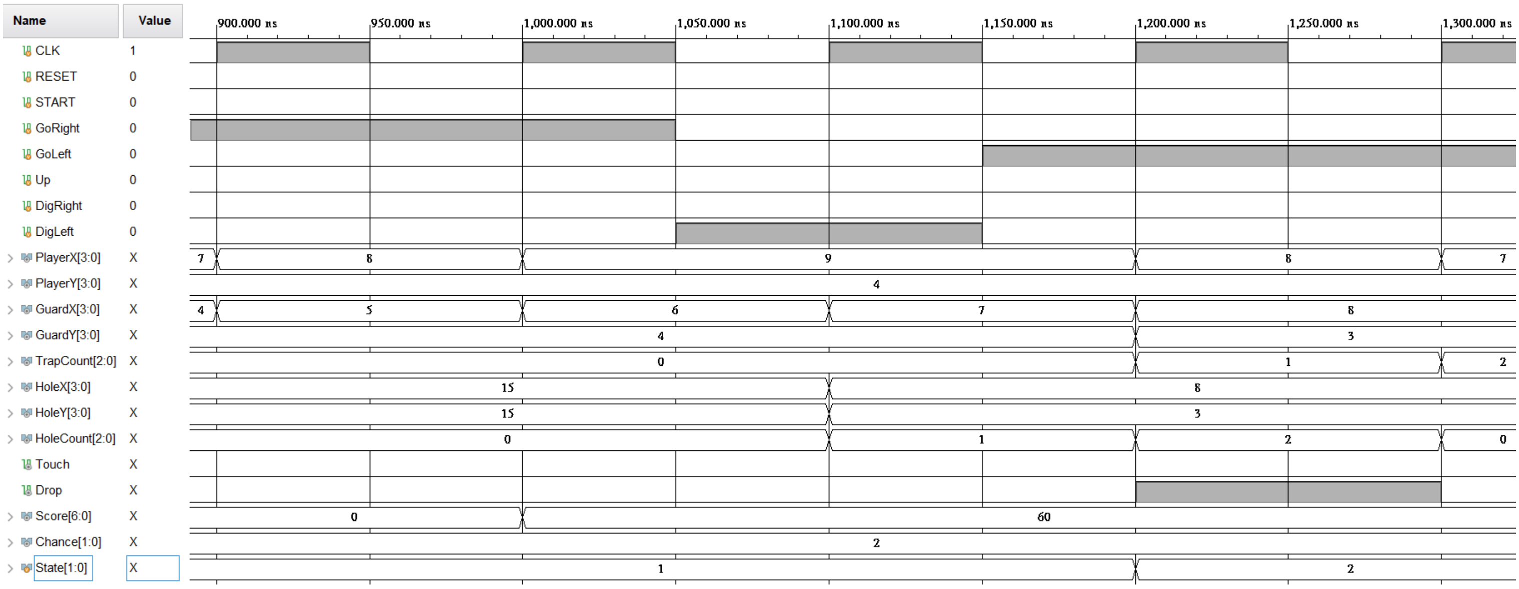
Behavior Simulation

(a) Successful exit

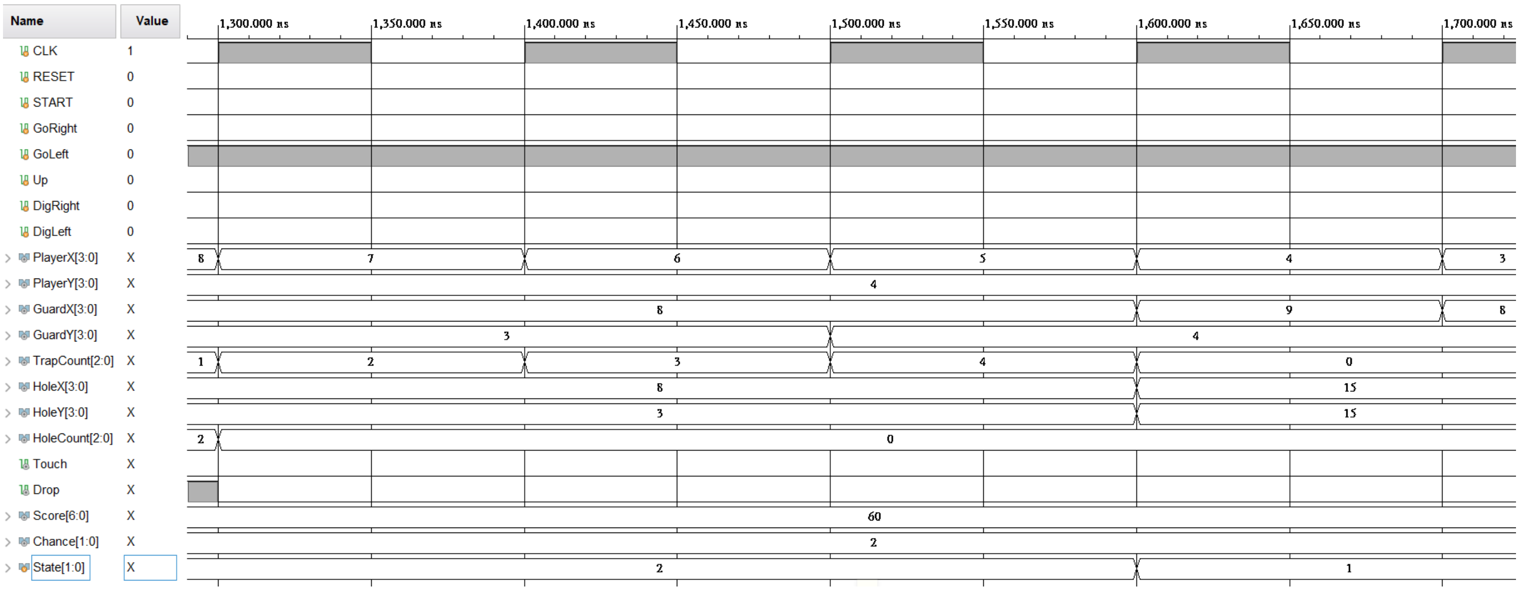




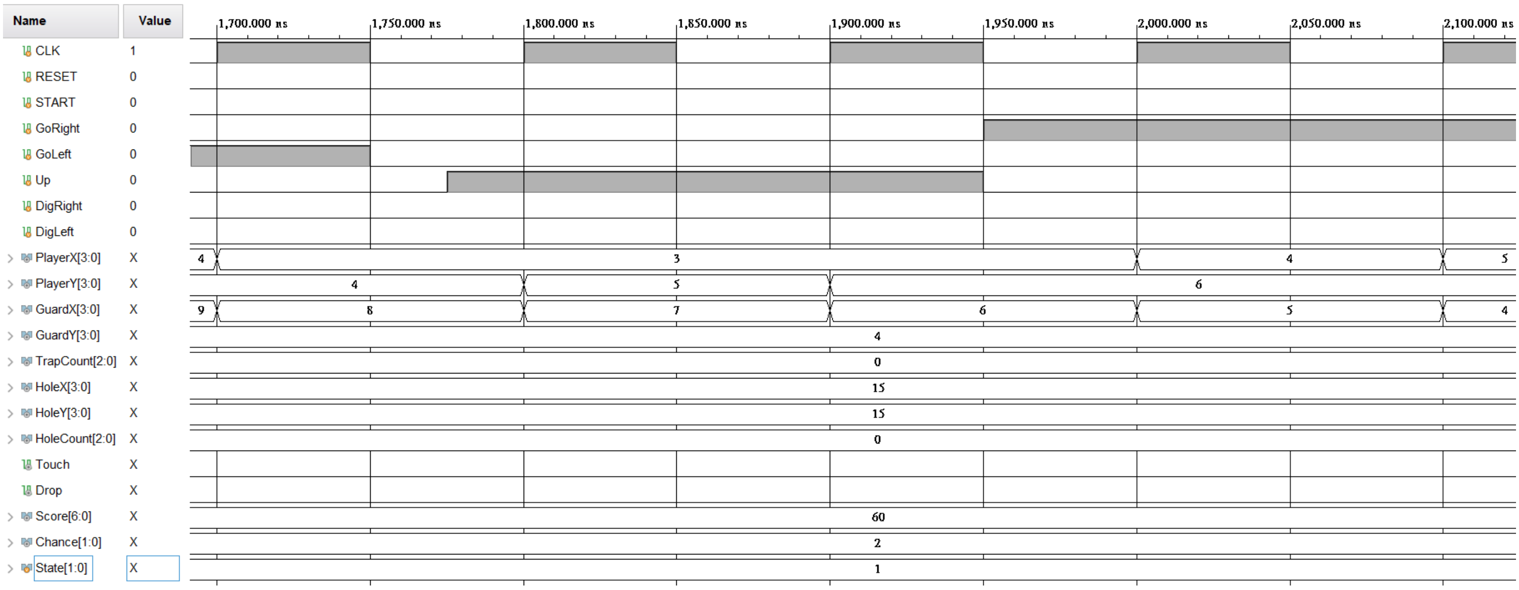
Name	Value
CLK	1
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	X
PlayerY[3:0]	X
GuardX[3:0]	X
GuardY[3:0]	X
TrapCount[2:0]	X
HoleX[3:0]	X
HoleY[3:0]	X
HoleCount[2:0]	X
Touch	X
Drop	X
Score[6:0]	X
Chance[1:0]	X
State[1:0]	X

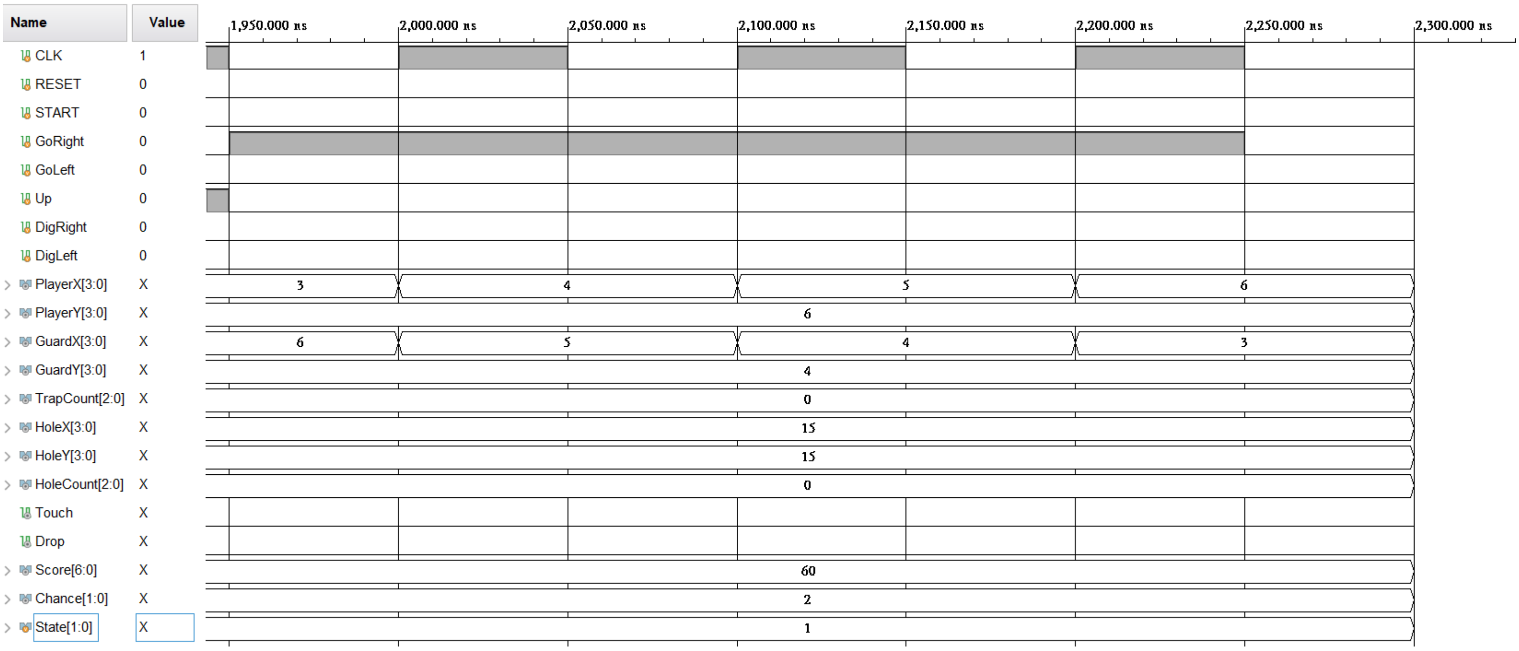


Name	Value
CLK	1
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	X
PlayerY[3:0]	X
GuardX[3:0]	X
GuardY[3:0]	X
TrapCount[2:0]	X
HoleX[3:0]	X
HoleY[3:0]	X
HoleCount[2:0]	X
Touch	X
Drop	X
Score[6:0]	X
Chance[1:0]	X
State[1:0]	X

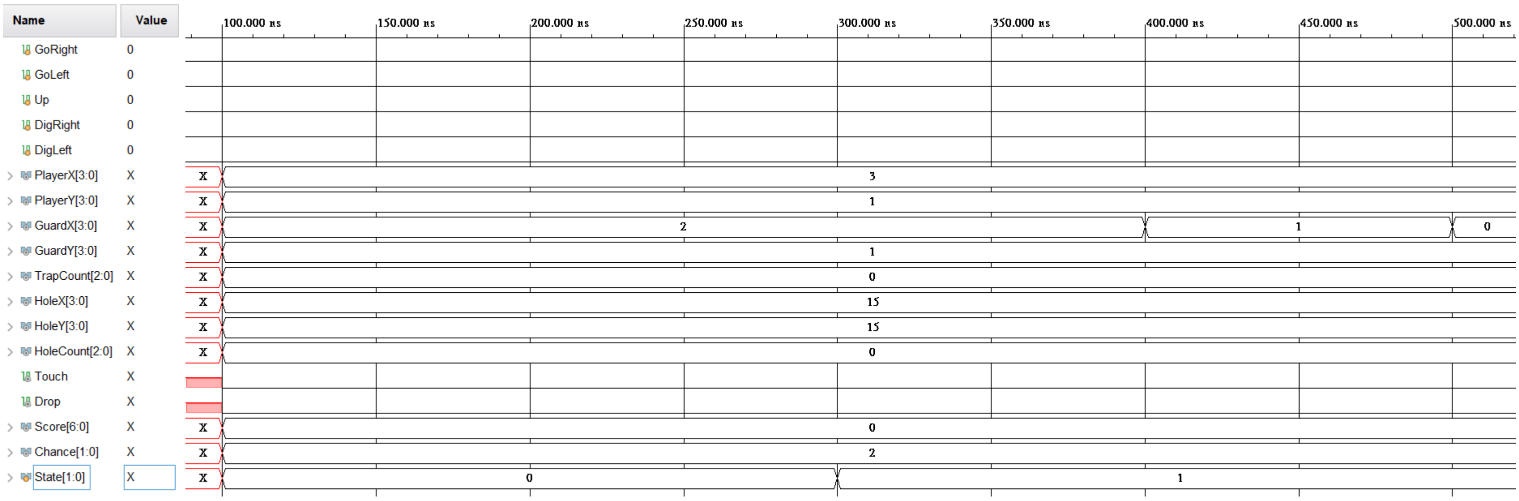


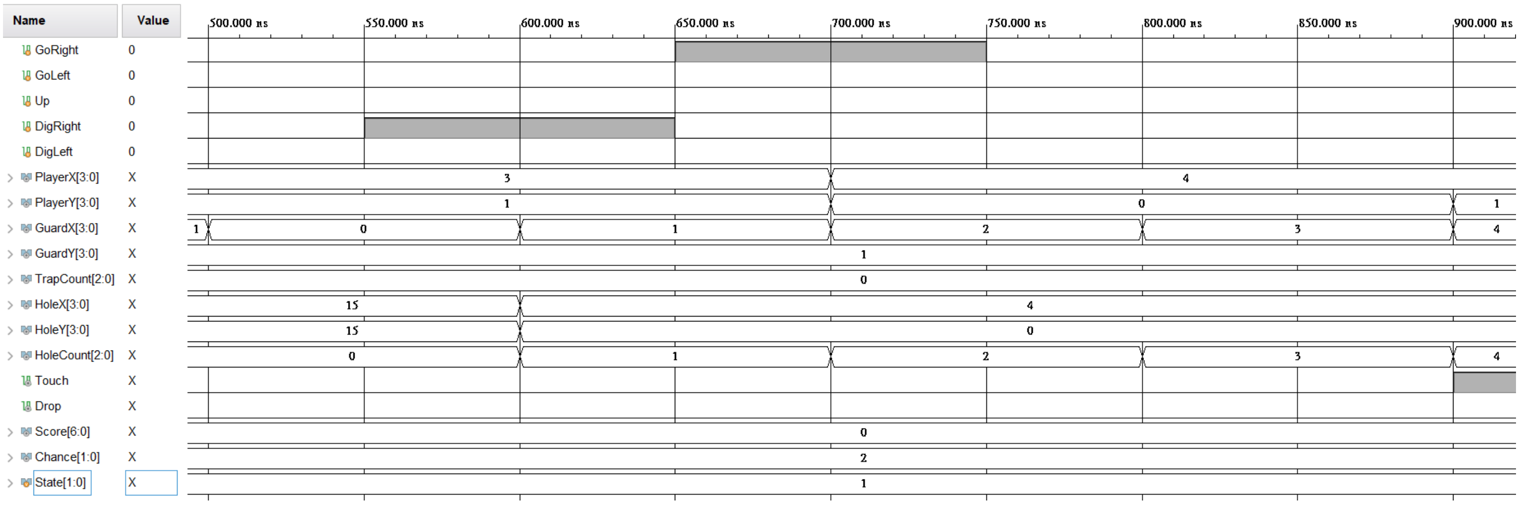
Name	Value
CLK	1
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	X
PlayerY[3:0]	X
GuardX[3:0]	X
GuardY[3:0]	X
TrapCount[2:0]	X
HoleX[3:0]	X
HoleY[3:0]	X
HoleCount[2:0]	X
Touch	X
Drop	X
Score[6:0]	X
Chance[1:0]	X
State[1:0]	X

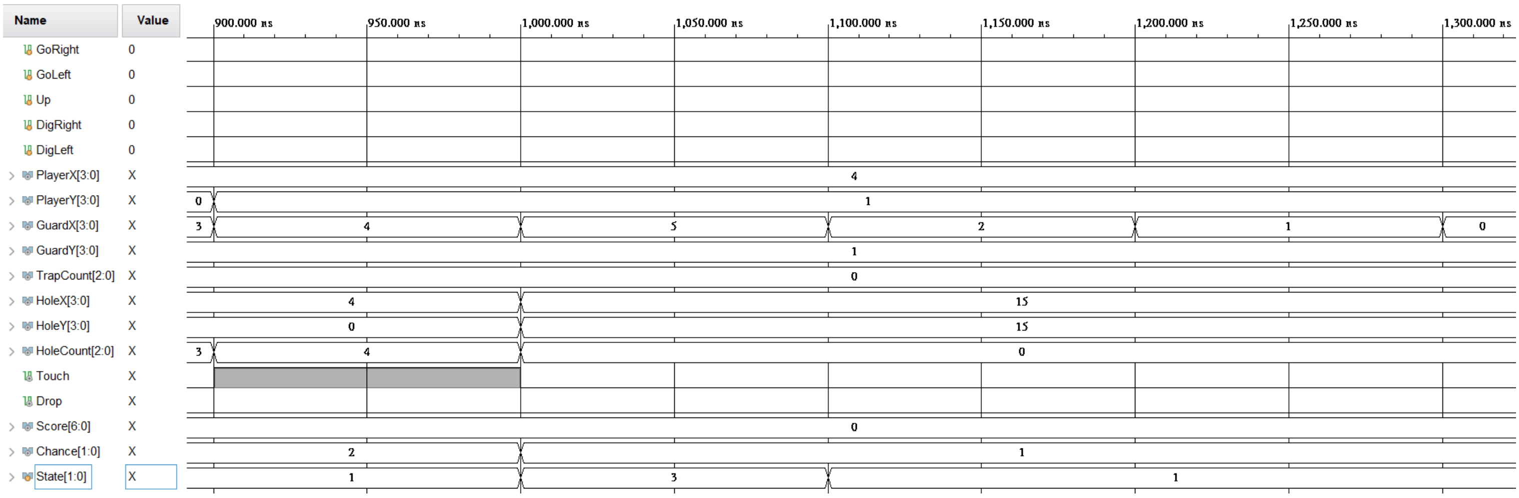

Name
Value
1,700.000 ns
1,750.000 ns
1,800.000 ns
1,850.000 ns
1,900.000 ns
1,950.000 ns
2,000.000 ns
2,050.000 ns
2,100.000 ns
CLK 1
RESET 0
START 0
GoRight 0
GoLeft 0
Up 0
DigRight 0
DigLeft 0
PlayerX[3:0] X 4 3 4 5
PlayerY[3:0] X 4 5 6
GuardX[3:0] X 9 8 7 6 5 4
GuardY[3:0] X 4
TrapCount[2:0] X 0
HoleX[3:0] X 15
HoleY[3:0] X 15
HoleCount[2:0] X 0
Touch X
Drop X
Score[6:0] X 60
Chance[1:0] X 2
State[1:0] X 1

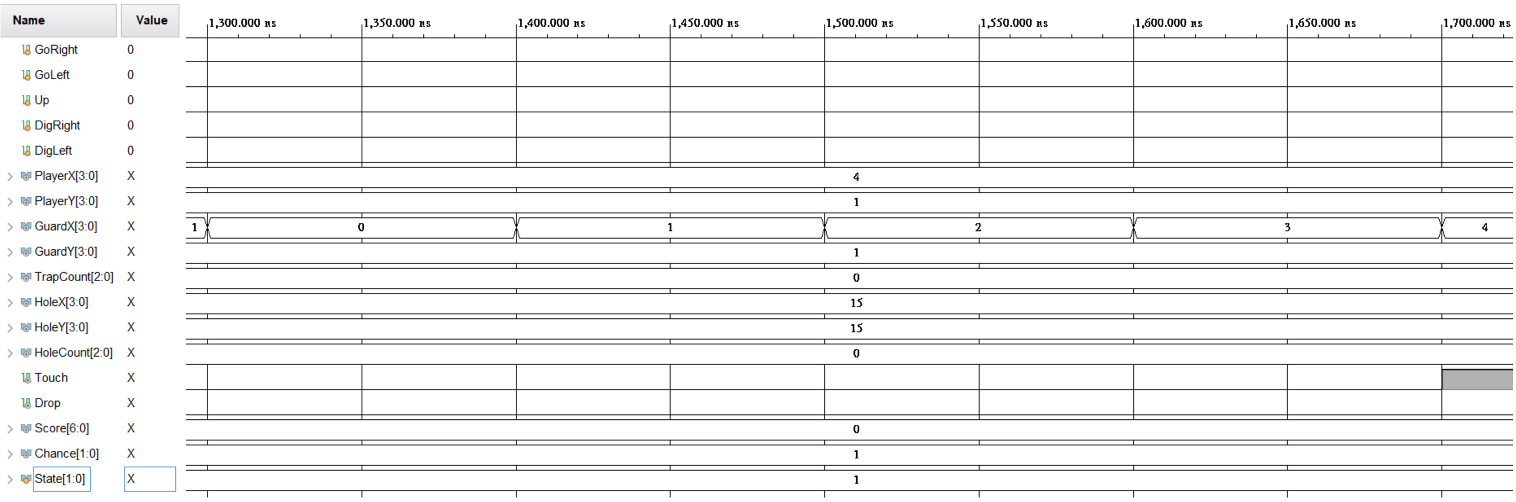

Name
Value
1,950.000 ns
2,000.000 ns
2,050.000 ns
2,100.000 ns
2,150.000 ns
2,200.000 ns
2,250.000 ns
2,300.000 ns
CLK 1
RESET 0
START 0
GoRight 0
GoLeft 0
Up 0
DigRight 0
DigLeft 0
PlayerX[3:0] X 3 4 5 6
PlayerY[3:0] X 6
GuardX[3:0] X 6 5 4 3
GuardY[3:0] X 4
TrapCount[2:0] X 0
HoleX[3:0] X 15
HoleY[3:0] X 15
HoleCount[2:0] X 0
Touch X
Drop X
Score[6:0] X 60
Chance[1:0] X 2
State[1:0] X 1

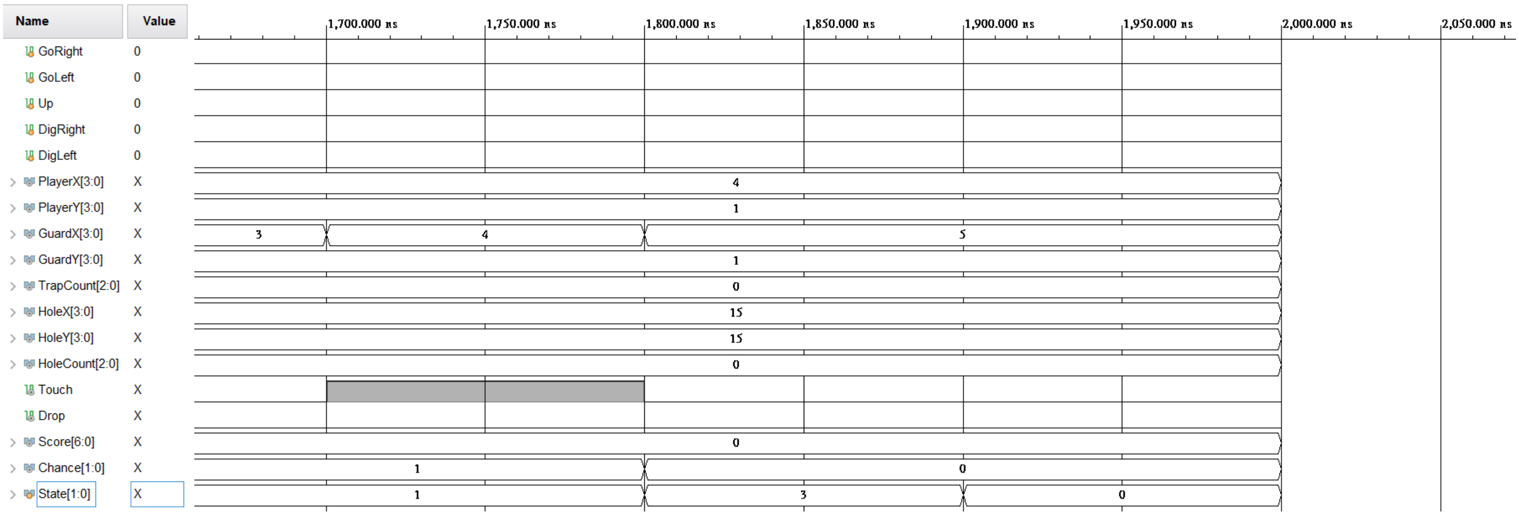
(b) fail cases twice





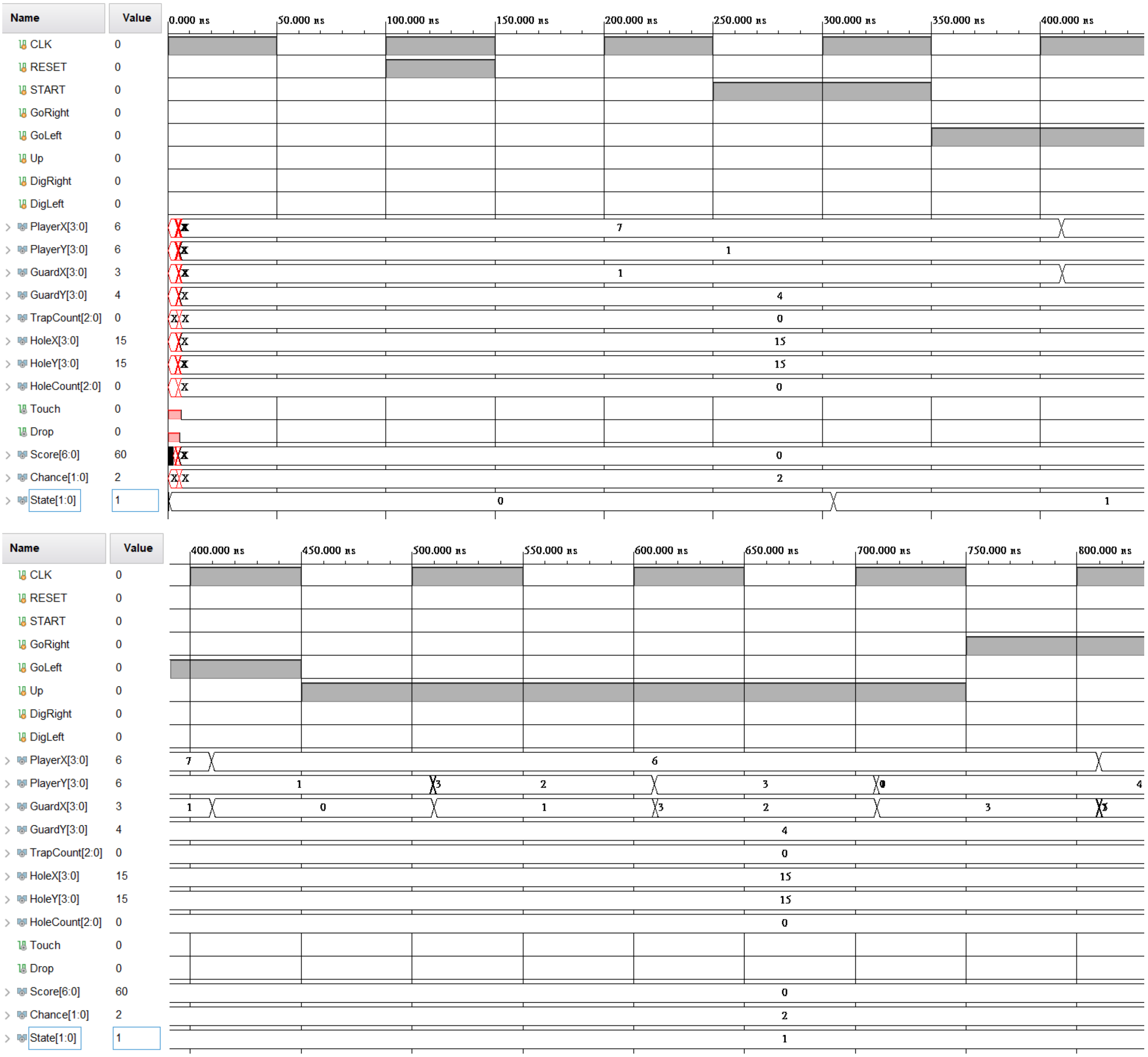


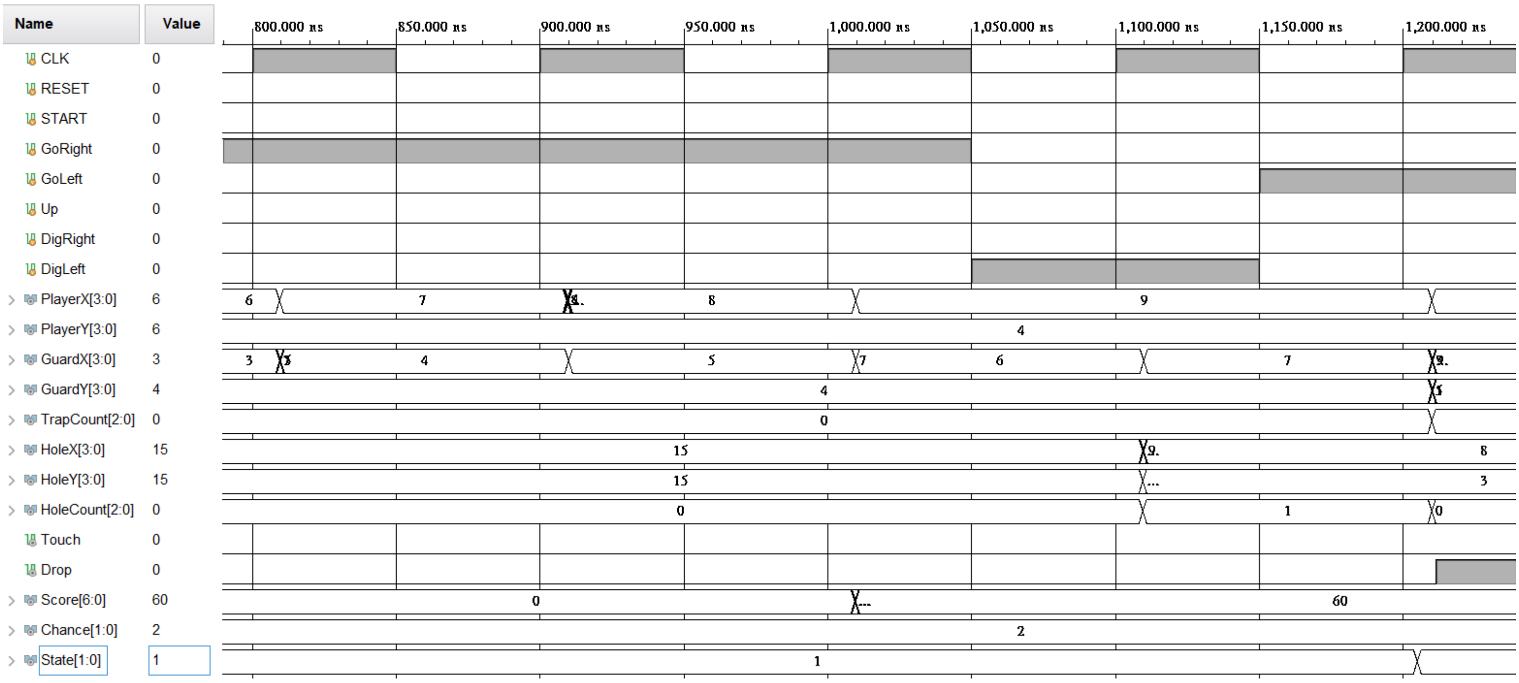

Name
Value
1,300.000 ns
1,350.000 ns
1,400.000 ns
1,450.000 ns
1,500.000 ns
1,550.000 ns
1,600.000 ns
1,650.000 ns
1,700.000 ns
GoRight 0
GoLeft 0
Up 0
DigRight 0
DigLeft 0
PlayerX[3:0] X 4
PlayerY[3:0] X 1
GuardX[3:0] X 1 0 1 2 3 4
GuardY[3:0] X 1
TrapCount[2:0] X 0
HoleX[3:0] X 15
HoleY[3:0] X 15
HoleCount[2:0] X 0
Touch X
Drop X
Score[6:0] X 0
Chance[1:0] X 1
State[1:0] X 1


Name
Value
1,700.000 ns
1,750.000 ns
1,800.000 ns
1,850.000 ns
1,900.000 ns
1,950.000 ns
2,000.000 ns
2,050.000 ns
GoRight 0
GoLeft 0
Up 0
DigRight 0
DigLeft 0
PlayerX[3:0] X 4
PlayerY[3:0] X 1
GuardX[3:0] X 3 4 5
GuardY[3:0] X 1
TrapCount[2:0] X 0
HoleX[3:0] X 15
HoleY[3:0] X 15
HoleCount[2:0] X 0
Touch X
Drop X
Score[6:0] X 0
Chance[1:0] X 1 0
State[1:0] X 1 3 0

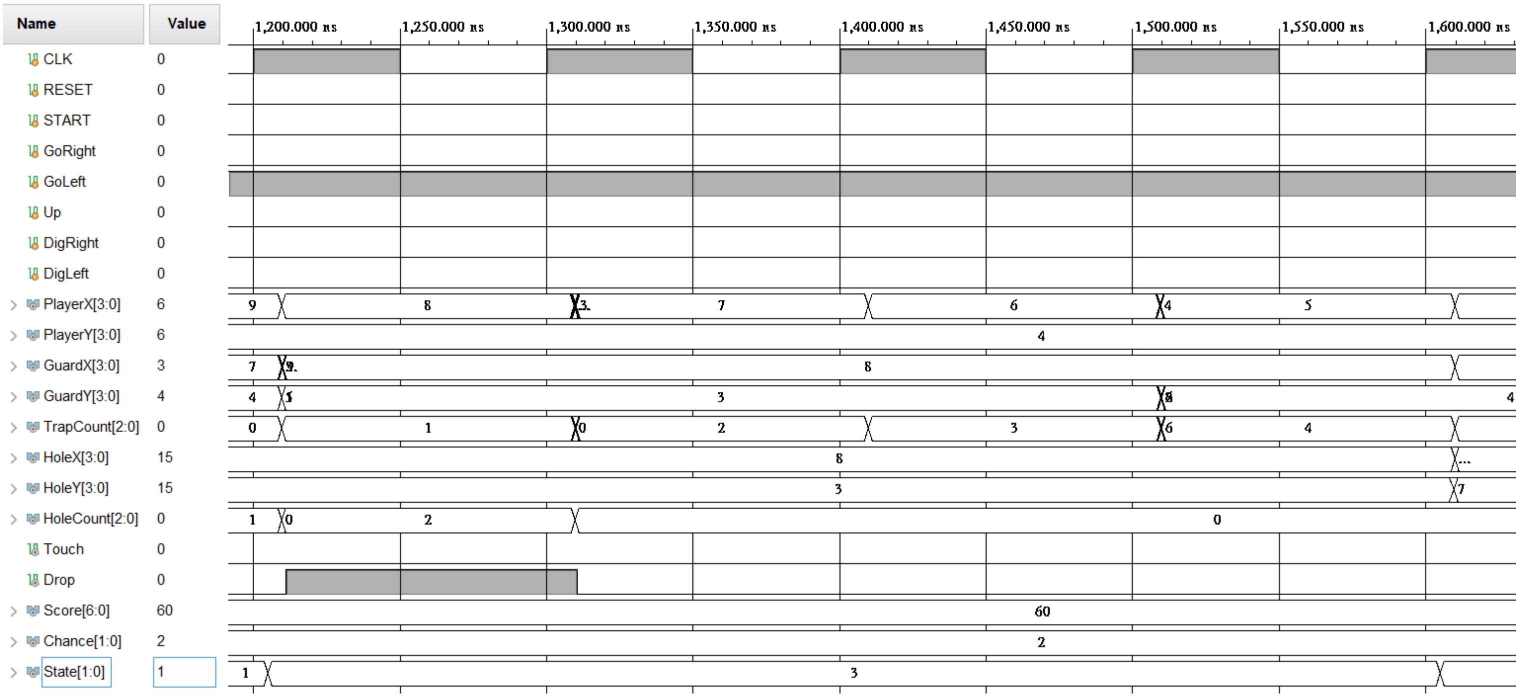
Post-route Simulation

(a) Successful exit

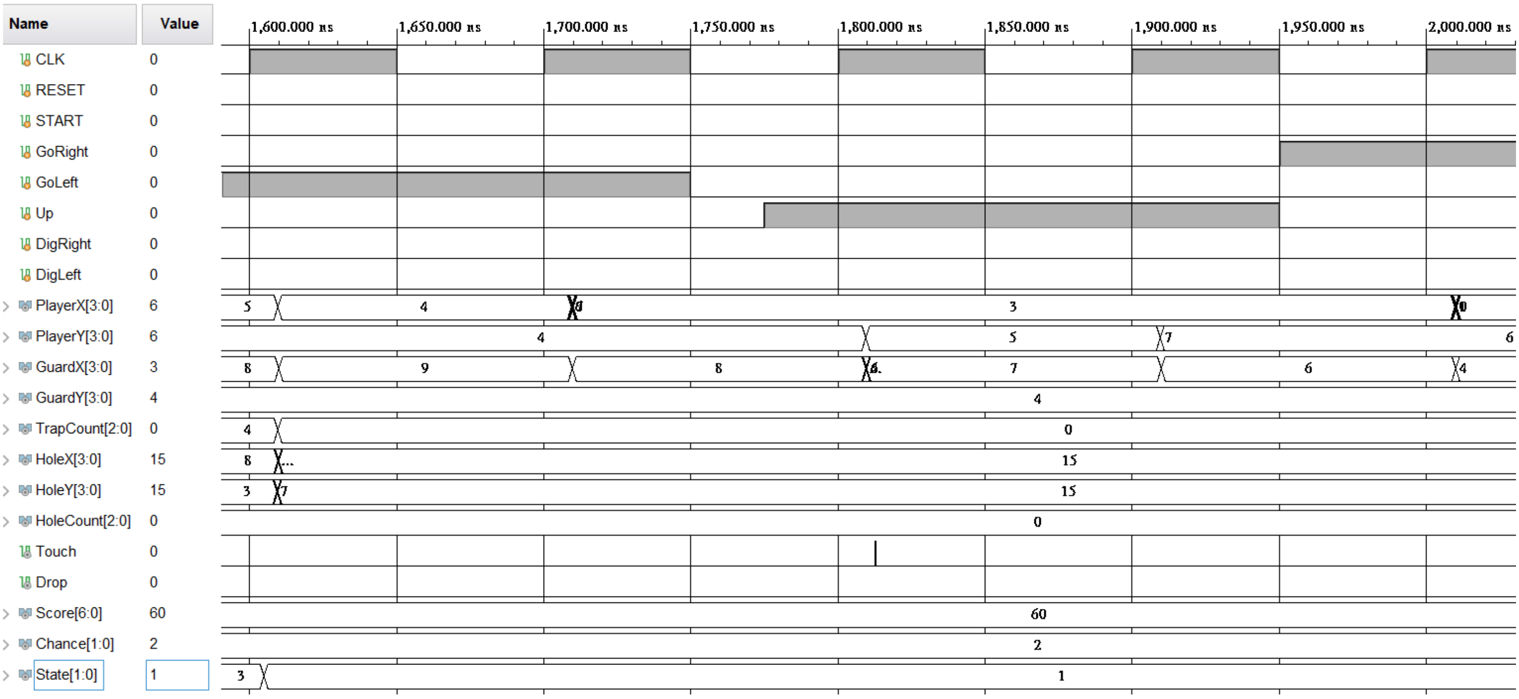




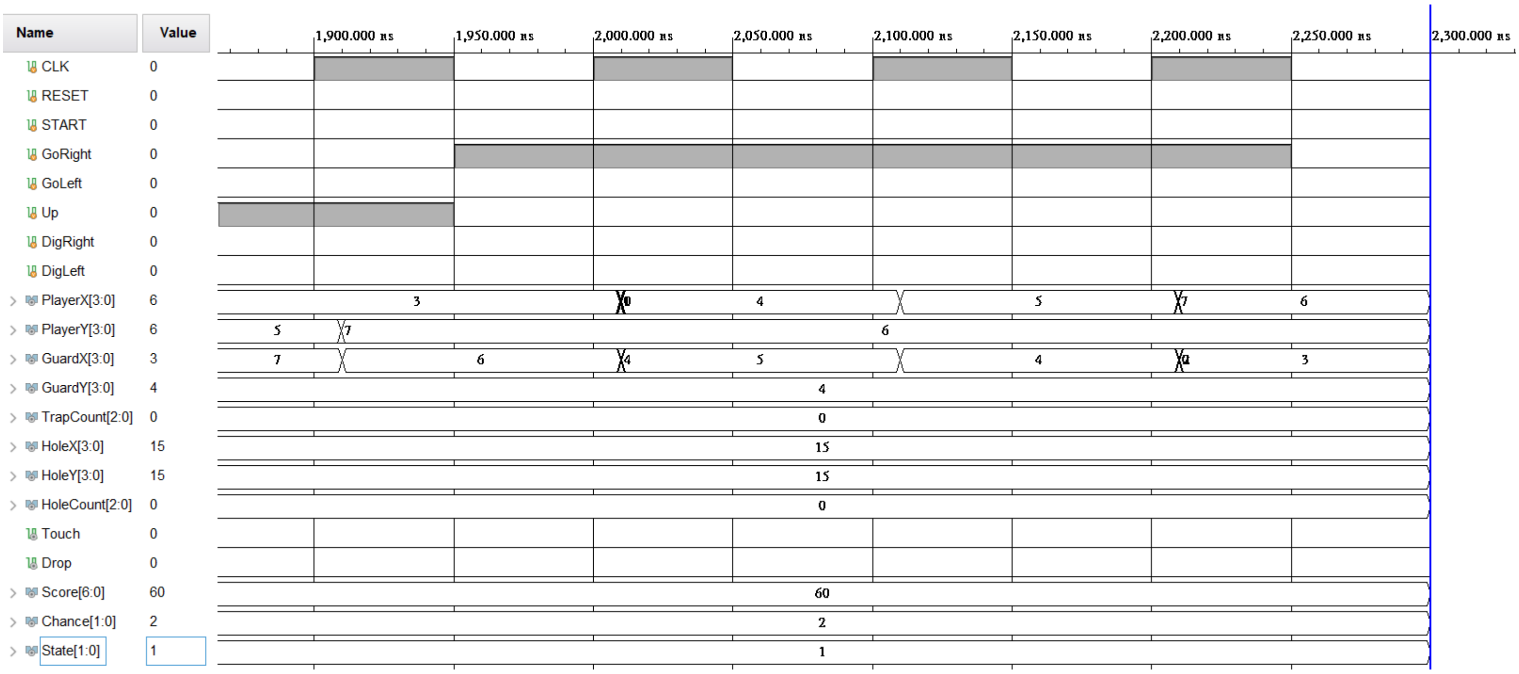
Name	Value
CLK	0
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	6
PlayerY[3:0]	6
GuardX[3:0]	3
GuardY[3:0]	4
TrapCount[2:0]	0
HoleX[3:0]	15
HoleY[3:0]	15
HoleCount[2:0]	0
Touch	0
Drop	0
Score[6:0]	60
Chance[1:0]	2
State[1:0]	1



Name	Value
CLK	0
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	6
PlayerY[3:0]	6
GuardX[3:0]	3
GuardY[3:0]	4
TrapCount[2:0]	0
HoleX[3:0]	15
HoleY[3:0]	15
HoleCount[2:0]	0
Touch	0
Drop	0
Score[6:0]	60
Chance[1:0]	2
State[1:0]	1

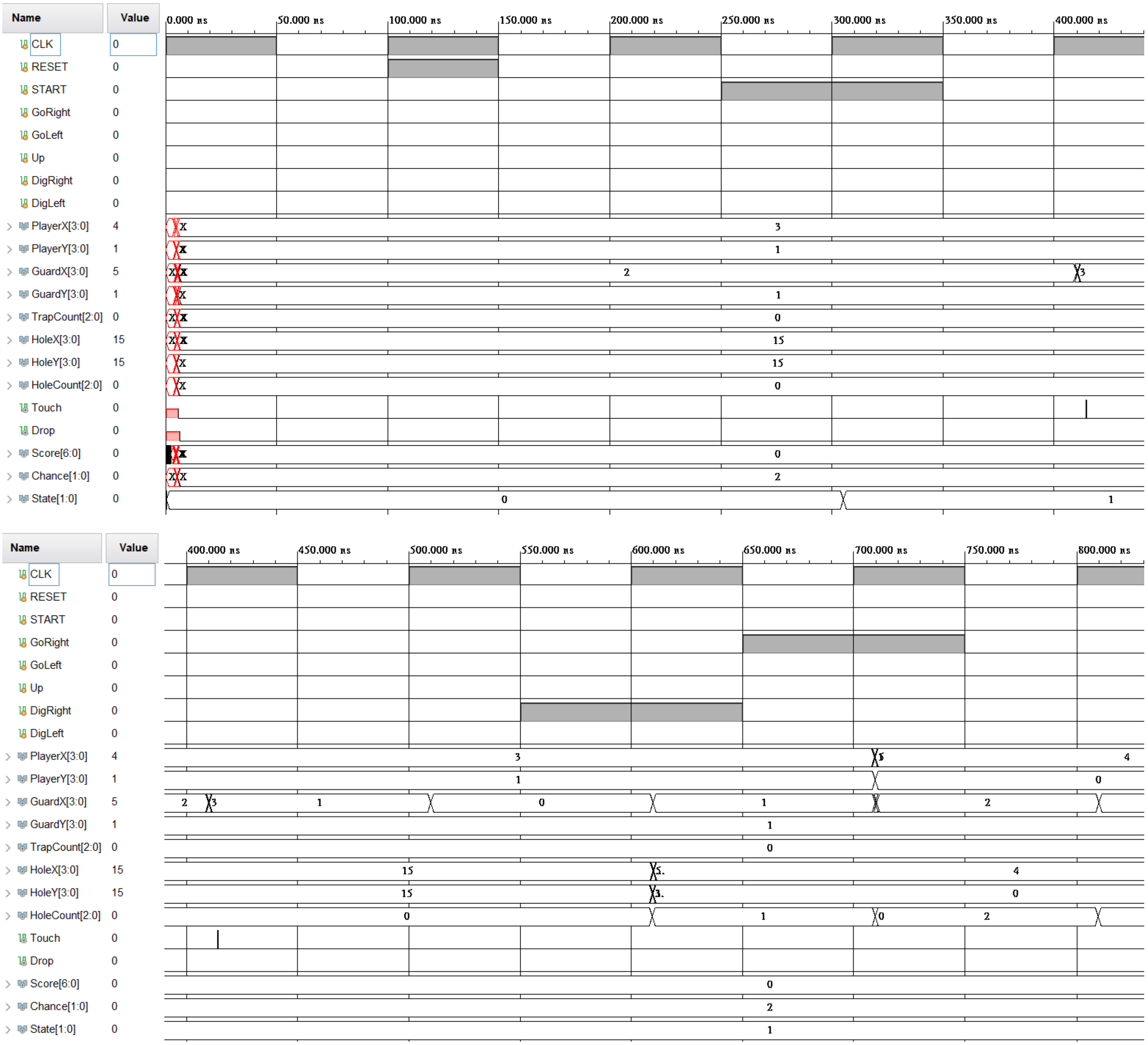


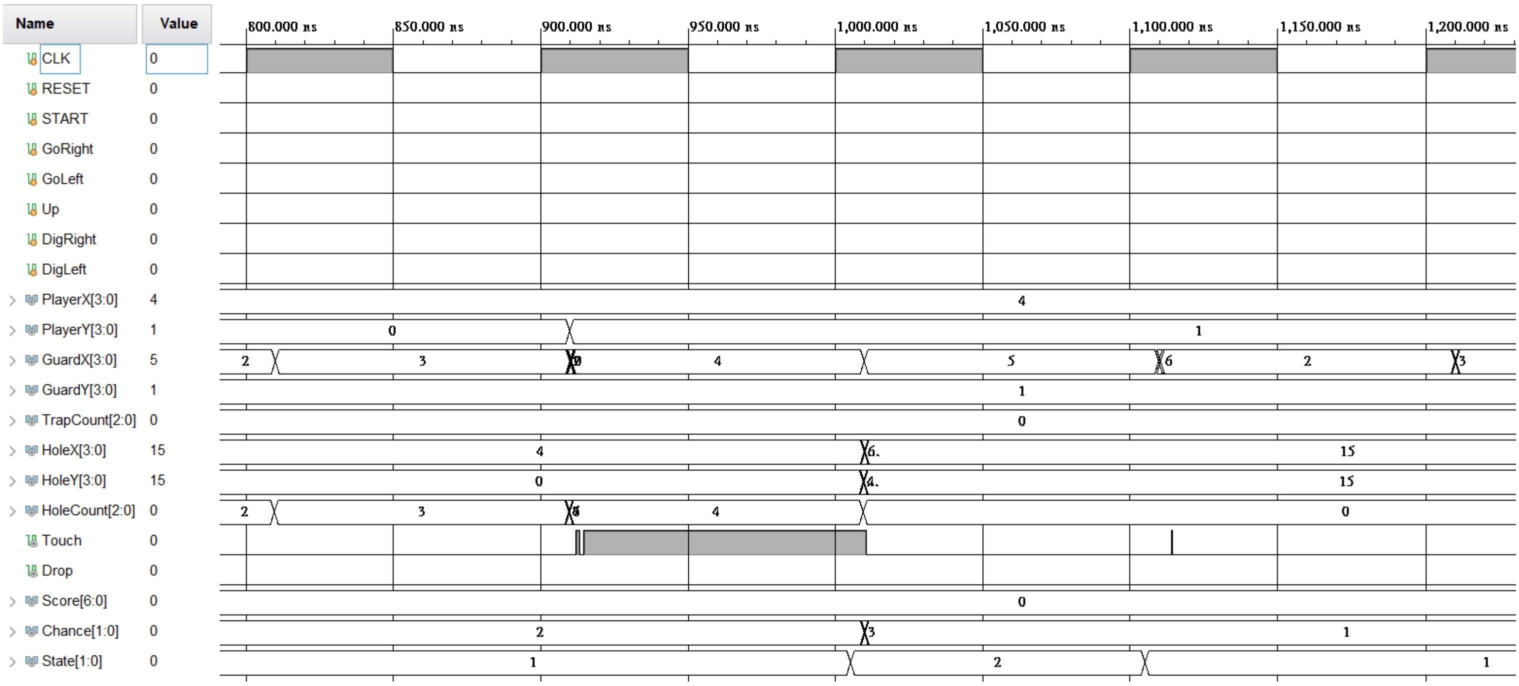
Name	Value
CLK	0
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	6
PlayerY[3:0]	6
GuardX[3:0]	3
GuardY[3:0]	4
TrapCount[2:0]	0
HoleX[3:0]	15
HoleY[3:0]	15
HoleCount[2:0]	0
Touch	0
Drop	0
Score[6:0]	60
Chance[1:0]	2
State[1:0]	1



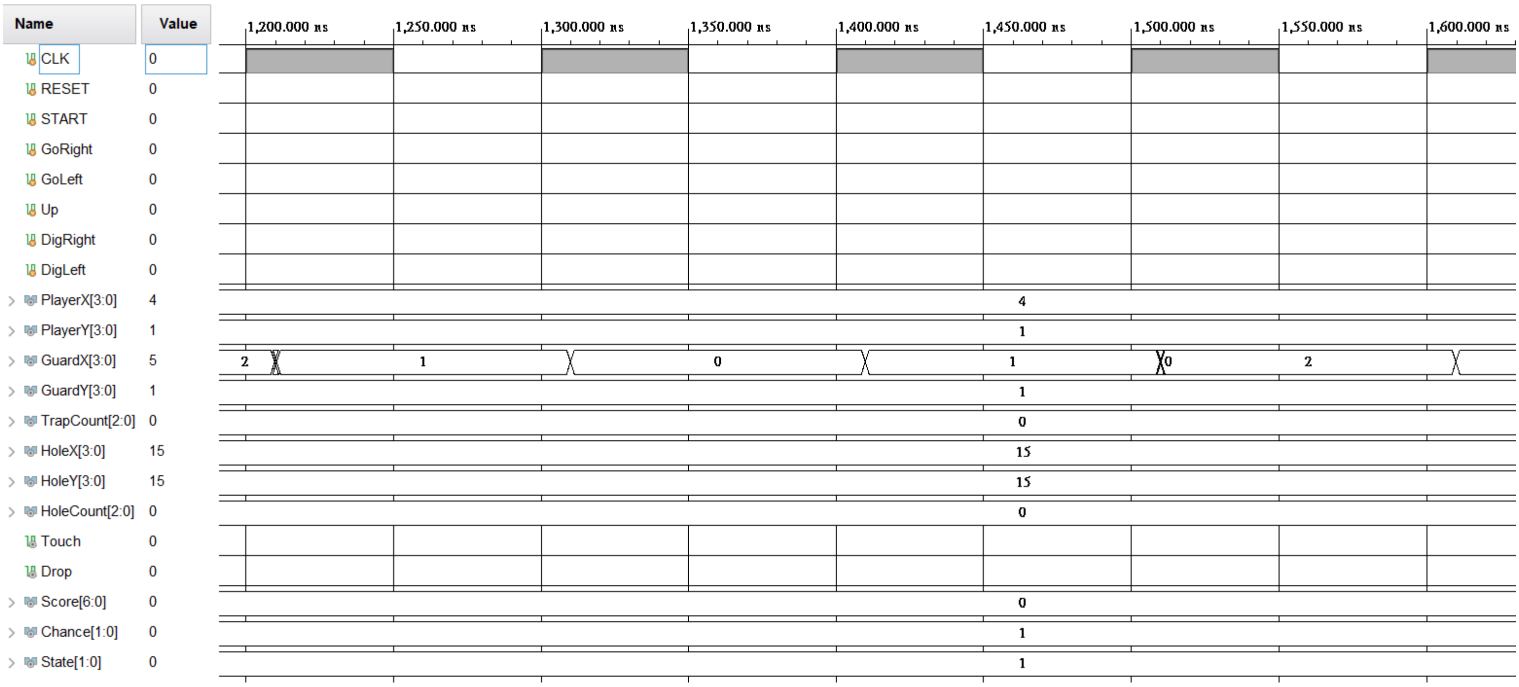
Name	Value
CLK	0
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	6
PlayerY[3:0]	6
GuardX[3:0]	3
GuardY[3:0]	4
TrapCount[2:0]	0
HoleX[3:0]	15
HoleY[3:0]	15
HoleCount[2:0]	0
Touch	0
Drop	0
Score[6:0]	60
Chance[1:0]	2
State[1:0]	1

(b) fail cases twice

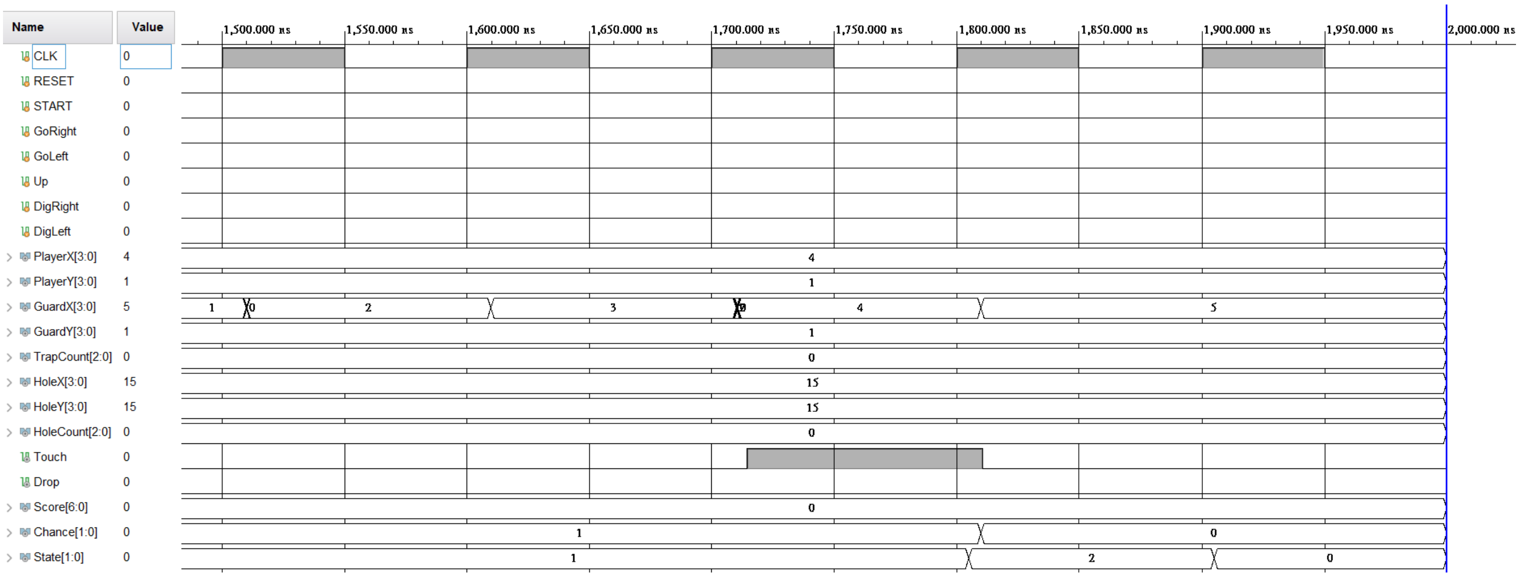




Name	Value
CLK	0
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	4
PlayerY[3:0]	1
GuardX[3:0]	5
GuardY[3:0]	1
TrapCount[2:0]	0
HoleX[3:0]	15
HoleY[3:0]	15
HoleCount[2:0]	0
Touch	0
Drop	0
Score[6:0]	0
Chance[1:0]	0
State[1:0]	0



Name	Value
CLK	0
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	4
PlayerY[3:0]	1
GuardX[3:0]	5
GuardY[3:0]	1
TrapCount[2:0]	0
HoleX[3:0]	15
HoleY[3:0]	15
HoleCount[2:0]	0
Touch	0
Drop	0
Score[6:0]	0
Chance[1:0]	0
State[1:0]	0



Name	Value
CLK	0
RESET	0
START	0
GoRight	0
GoLeft	0
Up	0
DigRight	0
DigLeft	0
PlayerX[3:0]	4
PlayerY[3:0]	1
GuardX[3:0]	5
GuardY[3:0]	1
TrapCount[2:0]	0
HoleX[3:0]	15
HoleY[3:0]	15
HoleCount[2:0]	0
Touch	0
Drop	0
Score[6:0]	0
Chance[1:0]	0
State[1:0]	0

Explanation

1. T = 100 [ns], Player = (3, 1), Guard = (2, 1) -> State = Stop, Chance = 2

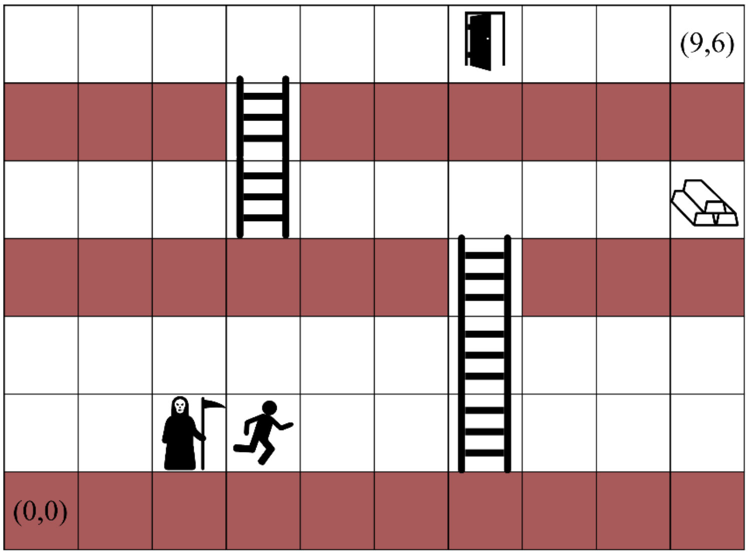


Fig.1 T = 100 [ns]

2. T = 350 [ns], State = Movement
3. T = 600 [ns], DigRight = 1 -> Hole = (4, 0)
4. T = 650 [ns], Player = (3, 1), Guard = (1, 1)

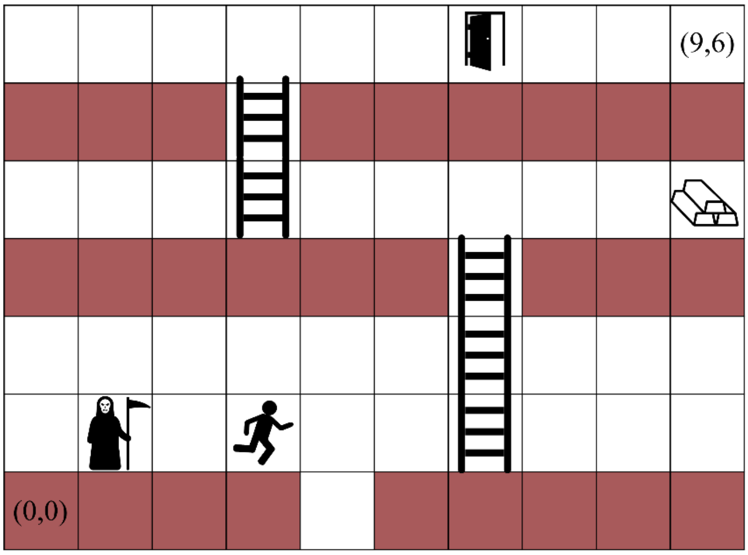


Fig.2 T = 650 [ns]

5. T = 700 [ns], GoRight = 1 -> Player = (4, 1)
6. T = 750 [ns], Player = (4, 0), Guard = (2, 1)

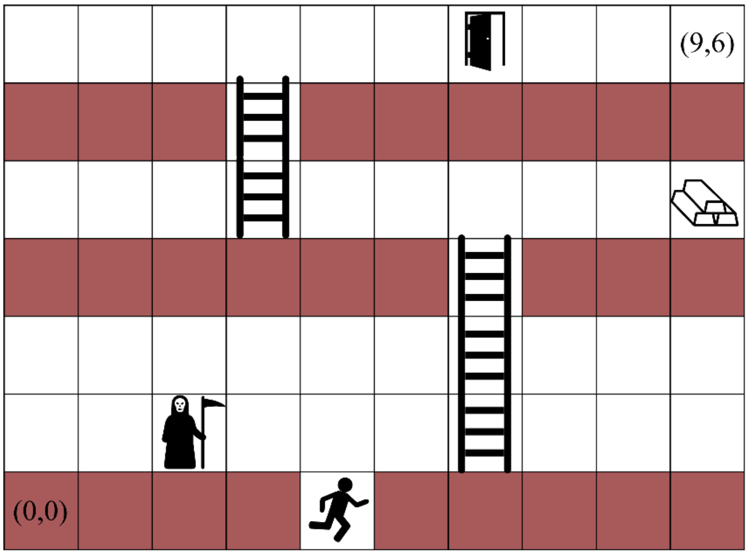


Fig.3 T = 750 [ns]

7. T = 950 [ns], HoleCount = 4, Touch = 1
8. T = 1050 [ns], Player = (4, 1), Guard = (4, 1) -> State = Die

9. T = 1150 [ns], State = Movement, Chance = 1

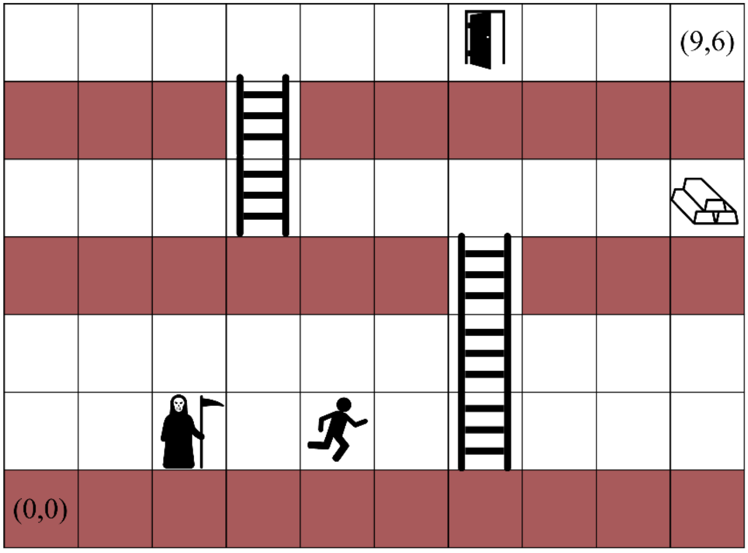


Fig.4 T = 1150 [ns]

10. T = 1750 [ns], Player = (4, 1), Guard = (4, 1) -> Touch = 1

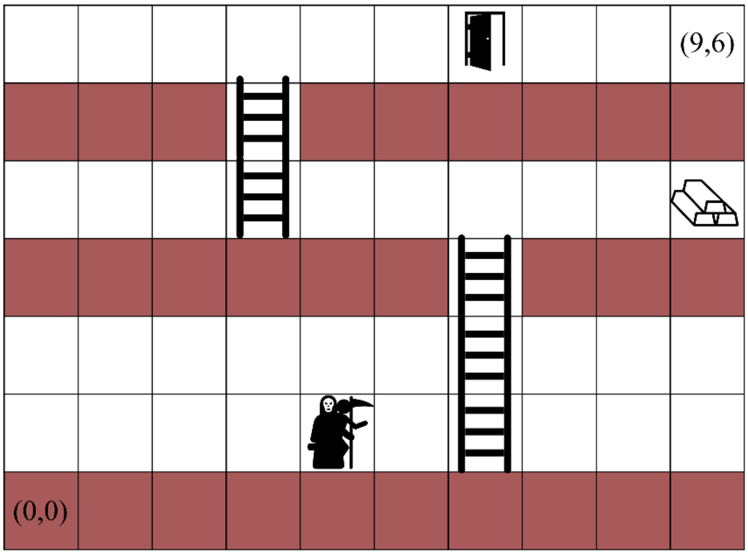


Fig.5 T = 1750 [ns]

11. T = 1850 [ns], State = Die, Chance = 0

12. T = 1950 [ns], State = Stop