CUT THE POWER - GAMEPLAY

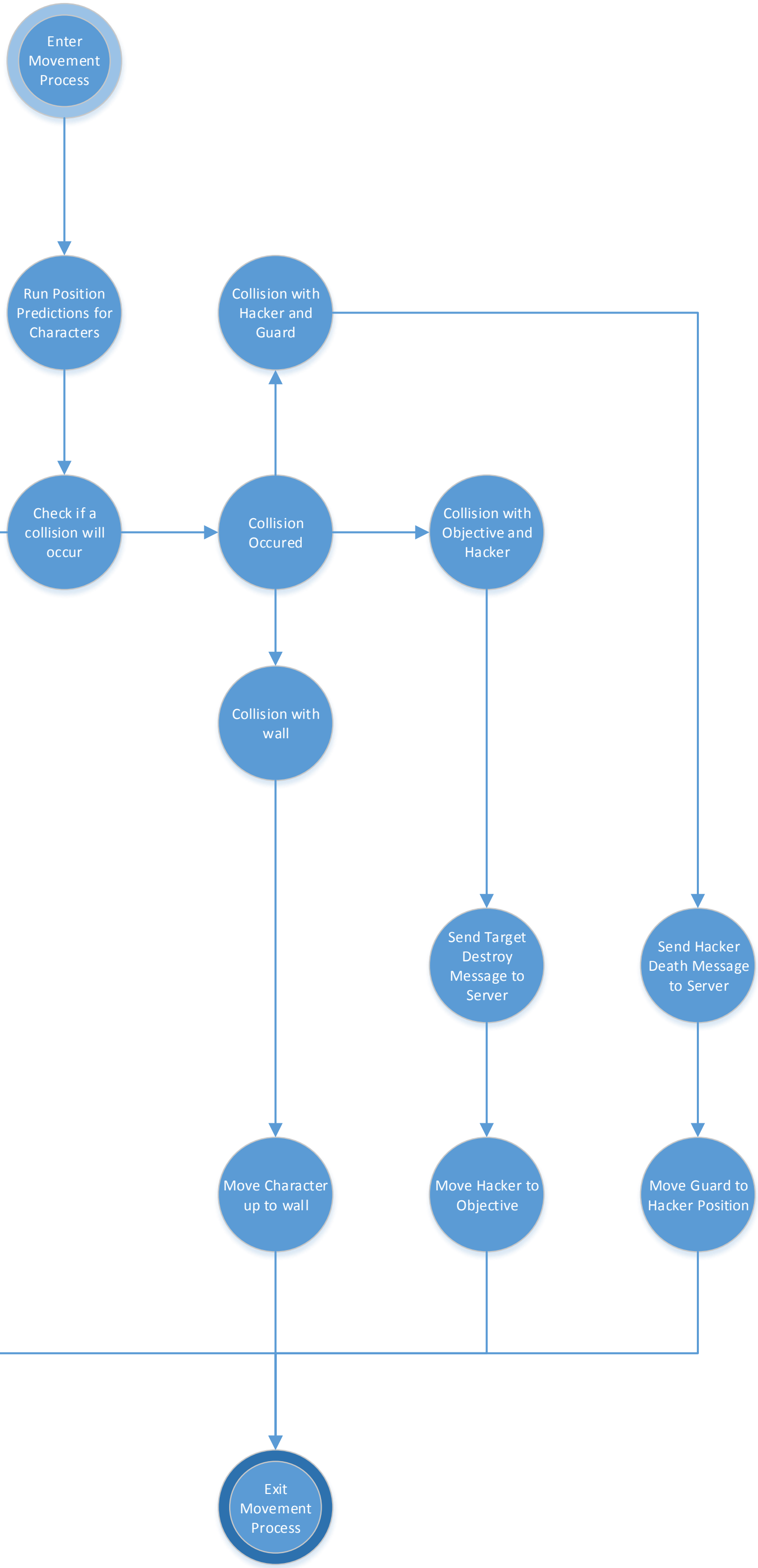
Cut the Power: A Data

Team members:

Joshua Campbell, Ian Davidson, Clark Allenby

Table of Contents

State Diagram:


Enter Movement Process
Run Position Predictions for Characters
Collision with Hacker and Guard
Check if a collision will occur
Collision Occured
Collision with Objective and Hacker
No collision
Collision with wall
Send Target Destroy Message to Server
Send Hacker Death Message to Server
Move Character to Desired Position
Move Character up to wall
Move Hacker to Objective
Move Guard to Hacker Position
Exit Movement Process

Pseudo Code:

```
Function MovementSystem(World)
    find the input entity

    //finding the controllable entity + processing
    for i = 0 to MAXENTITIES
        if world.mask[i] equals controllable entity mask
            create temporary entity
            assign actual controllable entity values to temporary entity

            if input == up
                temporary entity's y location -= temporary entity's velocity
            end if

            if input == down
                temporary entity's y location += temporary entity's velocity
            end if

            if input == left
                temporary entity's x location -= temporary entity's velocity
            end if

            if input == right
                temporary entity's x location += temporary entity's velocity
            end if

            if controllable entity has collision properties
                if Collision_System(World, Temporary Entity) equals no collision
                    actual entity gets assigned temporary entity's position data
                else if collision with stair
                    change floor
                else if collision with guard and actual entity is a hacker
                    kill hacker
                else if collision with hacker and actual entity is a guard
                    kill hacker
                else if collision with target and actual entity is a hacker
                    destroy target
                end if else
            else
                actual entity gets assigned temporary entity's position data
            end if else
        end if
    end for
end function


Function Collision_System(World, tempEntity)
```

```
        if wall_collision(world, tempEntity) equals true
                return Collision with Wall occurred
        end if

        if stair_collision(world, tempEntity) equals true
                return Collision with Stair occurred
        end if

        if guard_collision(world, tempEntity) equals true
                return collision with guard occurred
        end if

        if hacker_collision(world, tempEntity) equals true
                return collision with hacker occurred
        end if

        if target_collision(world, tempEntity) equals true
                return collision with target occurred
        end if

        return no collision
End Function

Function wall_collision(World, tempEntity)
        find tempEntity approximate position on the grid
        check world.level at tempentity approximate position to see if there is a wall
        if wall == true
                return collision occurred
        else
                return no collision
        end if
End Function

Function stair_collision(world, tempEntity)
        find tempEntity approximate position on the grid
        check world.level at tempentity approximate position to see if there is a stair
        if stair == true
                return collision occurred
        else
                return no collision
        end if
End Function
```

```
Function guard_collision(world, tempEntity)
    loop through all entities in world
        if world entity is a guard
            check to see if tempentity overlaps with found guard entity
            if collision occurred
                return collision occurred
            end if
        end if
    end loop
    return no collision occurred
End Function

Function hacker_collision(world, tempEntity)
    loop through all entities in world
        if world entity is a hacker
            check to see if tempentity overlaps with found hacker entity
            if collision occurred
                return collision occurred
            end if
        end if
    end loop
    return no collision occurred
End Function

Function target_collision(world, tempEntity)
    find tempEntity approximate position on the grid
    check world.level at tempentity approximate position to see if there is a target
    if target == true
        return collision occurred
    else
        return no collision
    end if
End Function
```

Milestones:

Milestone 1 – Base Systems and Components

The goal for February 21, 2014 includes, a basic map/player collision system, a player/player collision system, a basic movement system that simply moves the player to a new x and y coordinate based on input coming in, and designing the components needed to create these systems.

Milestone 2 – Extending the current systems and victory conditions

Due by **March 6, 2014.** This milestone will include creating randomized targets for hackers, a points/round system in order to create victory conditions, a system for handling victory conditions.

Milestone 3 – Extending the gameplay mechanics

Due by **March 20, 2014.** This milestone will include a predictive movement system in order to compensate for any lag that may occur between the client and the server. Other possible systems that may be developed in this milestone include: a basic AI system, traps, changing map elements (doors, pushing desks, cutting the power to a room or turning it back on, etc.).

Team Members' Responsibilities:

Joshua Campbell

- Designed and programmed the movement system
- Designed and programmed the collision system
- Designed and programmed the power-up system
- Programmed tag detection system
- Designed and programmed player creation tools
- Designed and programmed the map editor
- Implemented alternate character skins and theme songs into the game
- Programmed parts of the map loading function
- Designed and programmed the Frames Per Second scaling system that ensures that all players are moving at roughly the same speeds.
- Assisted Networking team with integrating client side networking functionality with the main gameplay loop.

Ian Davidson

- Ice tiles (currently not being used)
- Treadmill tiles(currently not being used)
- moveable/pushable objects(currently not being used)
- create stair function
- wormhole component
- some of the forces functions that got taken out
- all special "joke" skins except Clark, Abhishek, Josh, and Mat
- level design/creation
- gameplay design work
  - initial state transition diagram
  - headers for code

Clark Allenby

- Designed and programmed collision system
- Integrated gameplay code with networking
- Programmed chat system
- Designed and programmed movement system

Team Review:

Our team's task was to design the logic behind playing the game. The gameplay systems were very important as their functionality is one of the main things that determine if the game is usable and more importantly is fun. The first system we implemented was the movement system. After we got movement implemented, we needed a way for these moving entities to interact with other entities, so we designed and programmed the collision system. After these systems were created, we shifted our focus to integrating the client side networking code with our existing code. We also focused on integrating our new systems with the graphics team's maps (this consisted of using their maps to create abstract maps that our systems could process). During these integration phases, we encountered many issues such as needing to create a way to translate multiple tiles into some generic thing that could represent a wall or finding that due to latency, players could get stuck inside of other players. Towards the end of the project, our team branched into different paths such as assisting with creating the chat system, designing the power-up system, creating the FPS based scaling system, and creating alternate hidden skins for players.

Obstacles:

Out of the tasks we were assigned, our obstacles included creating dynamic systems that could easily accommodate new ideas for features, accommodating for latency issues, accommodating for different types of hardware, and creating smooth feeling movement and collisions so that the gameplay didn't feel choppy.