Skrímsli

Tyrique Campbell [†*], Oday Abushaban [†*], Heather Phillips [†*]
[*]University of Pittsburgh Undergraduate Student
[†] these authors contributed equally

I. The Game

This game is designed to be a co-operative, simultaneous, ”bop-it” with a central game board and 3 players. Each player wears a Gauntlet that has the capability to be programmed with one of three different Character Types: Knight, Mage, or Archer. Each character type comes with a Special Attack input to the system that exists on their Gauntlet.

II. The Physical Structure

This section will discuss the game's physical structure. Details on how this structure is utilized will be described in the following sections.

A. The Game Board

The main game board surface is comprised of a matrix of LEDs. The LEDs are each associated with a pair of buttons. On each player side, there is a button, four different colored and clearly labeled LEDs, and an IR receiver.

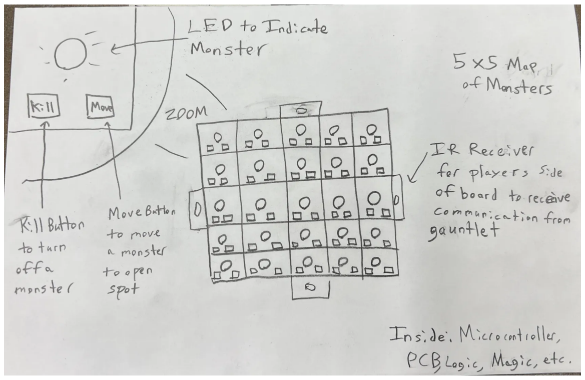

LED to Indicate Monster
Kill
Move
ZOOM
5x5 Map of Monsters
IR Receiver for players side of board to receive communication from gauntlet
Kill Button to turn off a monster
Move Button to move a monster to open spot
Inside: Microcontroller, PCB Logic, Magic, etc.

Fig. 1. Board Mock-Up

Within the main game board is a central MCU and the connecting circuitry. The game board is plug-in powered and has a power switch. There is a two-digit, seven-segment, display on the non-player side of the gameboard.

B. The Gauntlets

Each Gauntlet is powered by a lithium-ion battery and contains a button, an individual MCU, a patterned set of phototransistors, a spring-tensioned rod, an accelerometer, and an IR transmitter. The sensors are external to the Gauntlet structure and the MCU and associated circuitry is internal.

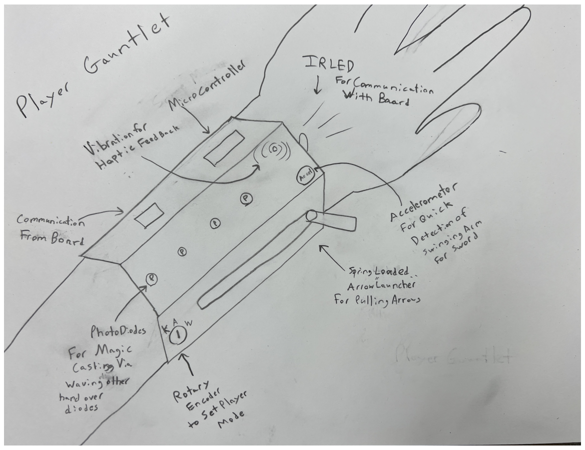

Player Gauntlet
MicroController
IRLED
For Communication With Board
Vibration for Haptic FeedBack
Accel
Communication From Board
Accelerometer For Quick Detection of Swinging Arm For Sword
Spring Loaded Arrow Launcher For Pulling Arrows
PhotoDiodes For Magic Casting Via Waving other hand over diodes
K A W
Rotary Encoder to Set Player Mode

Fig. 2. Gauntlet Mock-Up

III. Gameplay

This section will discuss the gameplay of the game. A block diagram of this gameplay is included in Figure 3.

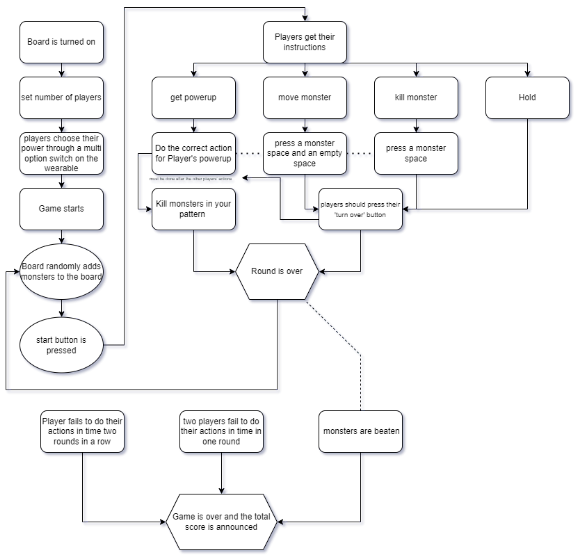

Board is turned on
set number of players
players choose their power through a multi option switch on the wearable
Game starts
Board randomly adds monsters to the board
start button is pressed
Players get their instructions
get powerup
move monster
kill monster
Hold
Do the correct action for Player's powerup
press a monster space and an empty space
press a monster space
must be done after the other players' actions
Kill monsters in your pattern
players should press their 'turn over' button
Round is over
Player fails to do their actions in time two rounds in a row
two players fail to do their actions in time in one round
monsters are beaten
Game is over and the total score is announced

Fig. 3. Block Diagram

A. Beginning the Game

At the beginning of the game, each player chooses which character type (Archer, Mage, Knight) they would like to play by positioning a rotary encoder on their individual Gauntlet. The game board must be switched On and then each player indicates that they are ready to play by pressing a clearly labeled button on their Gauntlet, ensuring their Gauntlet is facing the gameboard.

B. Game Rounds

Each round begins with the generation of Monsters on the game board, indicated through the lighting up of individual LEDs on the LED matrix, and the lighting up of one of three clearly labeled LEDs on each gameboard player side. The LEDs indicate the moves Move Monster, Destroy Monster, and Hold. The fourth LED may or may not begin to flash, indicating that a player has a Special Attack that they may use during this turn.

If a player has the Move Monster command, they must press two sequential LED-associated Move Monster buttons on the gameboard, indicating that a monster must move from the first location to the second. If they have the Destroy Monster command, they must press one Destroy Monster LED-associated button. If they have the Hold command, they must not press any buttons on the gameboard.

If a player has a flashing Special Attack indicator, they must perform their Special Attack, destroying multiple Monsters simultaneously. This must be done last and the round will end if the Special Attack is performed too quickly. The Special Attacks are performed through sensory inputs on the player Gauntlet and communicated to the main gameboard through IR transmission. The Special Attacks are listed below.

Special Attacks:

- Knight: Smite
  - Accelerometer input recognizing a swift downward motion of the player's hand
- Archer: Arrow
  - Spring-tensioned rod tracked by light-induced proximity sensors
- Mage: Fireball
  - Sequential phototransistor detection of hand/finger motion over gauntlet

C. Score Counting and Game Completion Conditions

At the end of each round, the central MCU within the gameboard accounts for the number of lit LEDs and awards players 1 point per killed Monster. If the wrong number of LEDs are lit at the end of a round, the players are penalized. If the players are off by one LED, they lose a point. If the players are off by two or more LEDs, they lose the game and their final score is shown on the seven-segment display on the gameboard.

The game may also end if players destroy all of the Monsters, if they are over-run by Monsters, or if a player fails to complete their move on two consecutive turns.

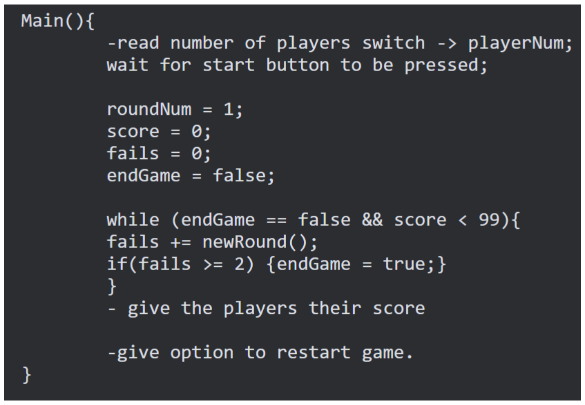
D. Additional Information

Player action command generation is statistically randomized with Move Monster occurring 2x more often than Destroy Monster, and Hold occurring 2x less often than Destroy Monster for each player. If all three players have successfully completed their action during the round preceding, one player is randomly chosen to complete a Special Attack.

The number of Monsters generated at the beginning of each round is normally distributed between 1 and a finite maximum that increases with each round.

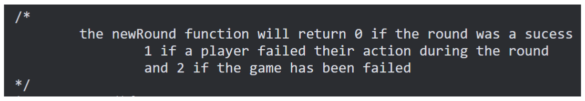
IV. Software

This section includes pseudo-code for the central MCU.



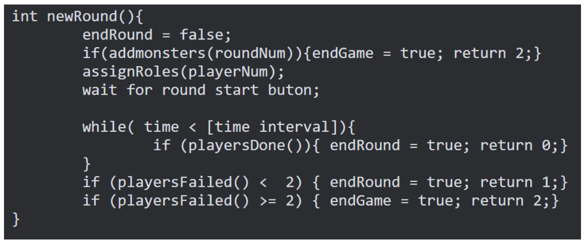
```
Main(){
        -read number of players switch -> playerNum;
        wait for start button to be pressed;

        roundNum = 1;
        score = 0;
        fails = 0;
        endGame = false;

        while (endGame == false && score < 99){
        fails += newRound();
        if(fails >= 2) {endGame = true;}
        }
        - give the players their score

        -give option to restart game.
}
```

Fig. 4. Main



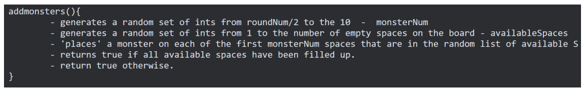
```
/*
        the newRound function will return 0 if the round was a sucess
                1 if a player failed their action during the round
                and 2 if the game has been failed
*/
```

Fig. 5. Comment



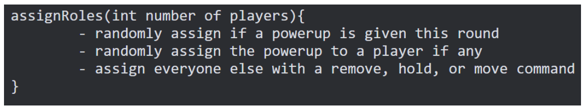
```
int newRound(){
        endRound = false;
        if(addmonsters(roundNum)){endGame = true; return 2;}
        assignRoles(playerNum);
        wait for round start buton;

        while( time < [time interval]){
                if (playersDone()){ endRound = true; return 0;}
        }
        if (playersFailed() <  2) { endRound = true; return 1;}
        if (playersFailed() >= 2) { endGame = true; return 2;}
}
```

Fig. 6. newRound



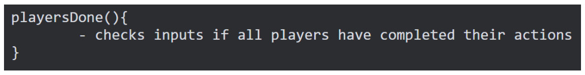
```
addmonsters(){
        - generates a random set of ints from roundNum/2 to the 10  -  monsterNum
        - generates a random set of ints from 1 to the number of empty spaces on the board - availableSpaces
        - 'places' a monster on each of the first monsterNum spaces that are in the random list of available S
        - returns true if all available spaces have been filled up.
        - return true otherwise.
}
```

Fig. 7. addMonsters



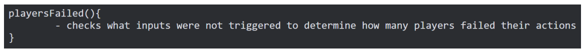
```
assignRoles(int number of players){
        - randomly assign if a powerup is given this round
        - randomly assign the powerup to a player if any
        - assign everyone else with a remove, hold, or move command
}
```

Fig. 8. assignRoles



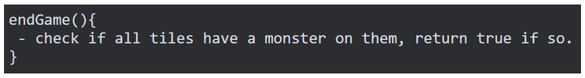
```
playersDone(){
        - checks inputs if all players have completed their actions
}
```

Fig. 9. playersDone



```
playersFailed(){
        - checks what inputs were not triggered to determine how many players failed their actions
}
```

Fig. 10. playersFailed



```
endGame(){
 - check if all tiles have a monster on them, return true if so.
}
```

Fig. 11. endGame

Note: if you have trouble reading the pseudo-code or any text within an image, open the .pdf in a new tab and zoom in.

V. Team Roles

The Team Roles for this project are as follows:

- Tyrique Campbell
  - Software Engineering Lead
- Oday Abushaban
  - Mechanical Engineering Lead
  - Electrical Engineer
- Heather Phillips
  - Wireless Communications Engineering Lead
  - Electrical Engineer