Project 3 Proposal

Project: Carltron

Authors: Sabastian Mugazambi, Kiet Tran, Derek Shang & Frederik Roenn Stensaeth

Repo URL: https://bitbucket.org/shangd7/carltron

User population: Gamers and Tron fans

The concert behind tron has been an interesting one since the release of the original 1982 Tron movie and gamers are always looking for versions of tron which are more challenging and has added features. Carltron is a version of tron with various features added to make the game more interesting. Tron fans and classic arcade gamers will be the main target population of Carltron.

Description:
Tron is a game based on a grid platform. Each player controls a car which is always in constant motion. The car leaves a trail which does not disappear but becomes a solid wall. Players control the cars in all four directions so as to avoid colliding with another player's path, their own path or the walls of the grid. The aim of the game is to force the opponent to collide before you do.This cane be done in many ways;

A player has limited chances to jump an opponent's path or their own path so as to avoid collision. Players can also instantaneously increase the speed of their car/bike by using turbo feature, thus creating a longer path in a short period of time and counter cornering their opponent. Each player is given three turbos at the beginning of a game. Players can pick up items that randomly appear on the grid so as to gain an advantage over their opponent.Last but not least we hope players will be able to gain invincibility for a limited time by picking up some of these items. The last man standing wins.

Feature List:

- Support of two players playing at the same time.
- Players can control the car to move in four directions on the grid.
- The car leaves an impassable path while traveling. Paths of different cars are displayed in different colors.
- A car dies on collision with another car, path or wall
- Support of "Best of 5" mode. A scoreboard is displayed on top of the grid.
- Ability to use turbos
- Extra turbos (represented by a dot) are regenerated from time to time and can possibly appear anywhere on the grid. Players pass these dots to gain additional turbos.
- Ability to jump
- Ability to clear grid
- Players can become invulnerable for 5 seconds, but will not leave a path behind.
- An option menu that lets players adjust the general speed of cars.

- 1 player mode with no AI.
- 1 player mode with AI.
- Players are able to adjust the difficulty of AI.

Deliverables:
The structure of our code will be based on Model-View-Controller pattern. The model of the game consists of three files: Grid.java, LightCycle.java and Bonus.java. Grid represents the map of the game, which defines important game parameters such as boundaries and paths. LightCycle is a class of objects that represent the vehicle each player uses in game. Bonus is an abstract class that defines common properties that all kinds of in-game bonuses share. Specific bonus types inherit from this class and define additional properties. The view of this game is mostly FXML files, which present the GUI of the game. Finally, we have another file GameManager.java that defines the controller of this game.

Class Main

- public void start(Stage primaryStage)

  Loads the constructor and sets up the stage/ scene.
- public static void main(String[] args)

  Backup method when start() does not work.

Class GameManager implements EventHandler<KeyEvent>

GameManager is out controller class. The class handles input in the form of keystrokes or button clicks and decides on what to do with them - whether it be update the position of the LightCycles, pause/resume the game, quit the program, or ignore the input.

- public void handle(KeyEvent keyEvent)
  - Analyzes the keystroke that occurred and updates the position of the LightCycle. Does nothing if the keystroke was not one that controls either of the lightCycles. Takes a KeyEvent as input. Returns nothing.
- public void onPauseButton(ActionEvent actionEvent)
  - Pauses/ resumes the game depending on what state the game is in. If game is being resumed the method sets up the animation timer, and if it being paused the timer is cancelled. Takes an ActionEvent as input. Returns nothing. Side-effects include that the state of that game, and subsequently the window, changes.
- public void updateAnimation()
  - Gets the position of the LightCycles and updates it depending on what direction the LightCycles are going in. Checks whether either of the LightCycles have crashed. updateAnimation() does not take any input and does not return anything either. Side-effects include changes in the position of the LightCycles.
- public void setUpAnimationTimer()
  - Sets up the animation timer. This timer serves to let us know when we need to update our animation.
- public void initialize()

- Calls setUpAnimationTimer() to set up the animation timer. Requires no input and the method returns nothing.

Class Grid (model)

- Instance variables:
  private static final int WIDTH
  private static final int HEIGHT
  private HashMap<Point, Path> collisions
- private Class Path:
  private LightCycle player
  public Path(LightCycle player) (constructor)
  public void getPlayer()
- public Grid(int width, int height) (constructor)
  The constructor for the grid class. Takes a width and a height, both as integers, and creates a grid that will be used to store the paths created by the lightCycles.
- public int getWidth()
  Returns the width of the grid as an integer.
- public int getHeight()
  Returns the height of the grid as an integer.
- public void setWidth(int width)
  Sets the width of the grid to a certain size.
- public void setHeight(int height)
  Sets the height of the grid to a certain size.
- public void clearAll()
  Removes all the paths stored in the grid.

Class LightCycle (model)

- Instance variables:
  private int velocity X (= -1 || 0 || 1 || -2 || 2)
  private int velocity Y (= -1 || 0 || 1 || -2 || 2)
  private List<Bonus> bonuses
  private Color color
  private int life
- public LightCycle(int life, Color color) (constructor)
  The constructor for the LightCycle class. Takes an integer that represents the amount of lives that the LightCycle is to have and a color object to represent the color of the path created by the LightCycle.
- public int getVelocityX()
  Returns the velocity in the x-direction of the LightCycle.
- public int getVelocityY()
  Returns the velocity in the y-direction of the LightCycle.
- public int getLife()
  Returns the integer representing how many lives the player has left.
- public Color getColor()

Returns the color of the LightCycle. This color will be used to distinguish between the paths of different LightCycles.

- public List<Bonus> getBonuses()
  Returns a list of length three of bonuses that the LightCycles has. The first element is turbo, the second element is jump, and the third is protector.
- public void setVelocityX(int newVelocity)
  Sets the x-velocity of the LightCycle to be a given value, namely newVelocity.
- public void setVelocityY(int newVelocity)
  Sets the y-velocity of the LightCycle to be a given value, namely newVelocity.
- public void setLife(int newLife)
  Sets the number of lives that the LightCycle has to a new value, newLife.
- public void step()
  Updates position of the LightCycle. This position will be different from before because the LightCycle is always in motion.
- public void consume(String type)
  Activates one of the bonuses of the LightCycle and decrements the number of that type of bonus the LightCycle has available. Activating the bonus alter the behavior of the LightCycle.

Abstract Class Bonus (model)

- Instance variables:
  private int amount
- public Bonus() (constructor)
  The constructor for the bonus class. Takes no input.
- abstract public void consume()
  Declares the method, which should be implemented by every class that inherits from Bonus.
- public int getAmount()
  Returns the amount of the bonus.
- public void setAmount(int amount)
  Sets the amount of the bonus.

Class Turbo extends Bonus

- public Turbo(int newAmount) (constructor)
  The constructor for the turbo class. Takes an integer to represent the amount of turbos available to the LightCycle and sets amount to be this number.
- public void consume()
  Consumes a potion of turbo and gains a burst of speed over a short time of duration.

Class Jump extends Bonus

- public Jump(int newAmount) (constructor)

The constructor for the jump class. Takes an integer to represent the amount of turbos available (newAmount) to the LightCycle and sets amount to be this number.

- public void consume()

  Consumes a potion of jump and gains the ability to cross a path and leave no path behind while the potion is in effect.

Class Protector extends Bonus

- public Protector(int newAmount) (constructor)

  The constructor for the protector class. Takes an integer to represent the amount of turbos available (newAmount) to the LightCycle and sets amount to be this number.

- public void consume()

  Consumes a potion of Protector and gains the ability to become invincible for a short duration of time.

fxml/carltron-menu.fxml (view)
fxml/carltron-rules.fxml (view)
fxml/carltron-winner.fxml (view)
fxml/carltron-game.fxml (view)

Tests:
Files that we are going to write unit tests for:

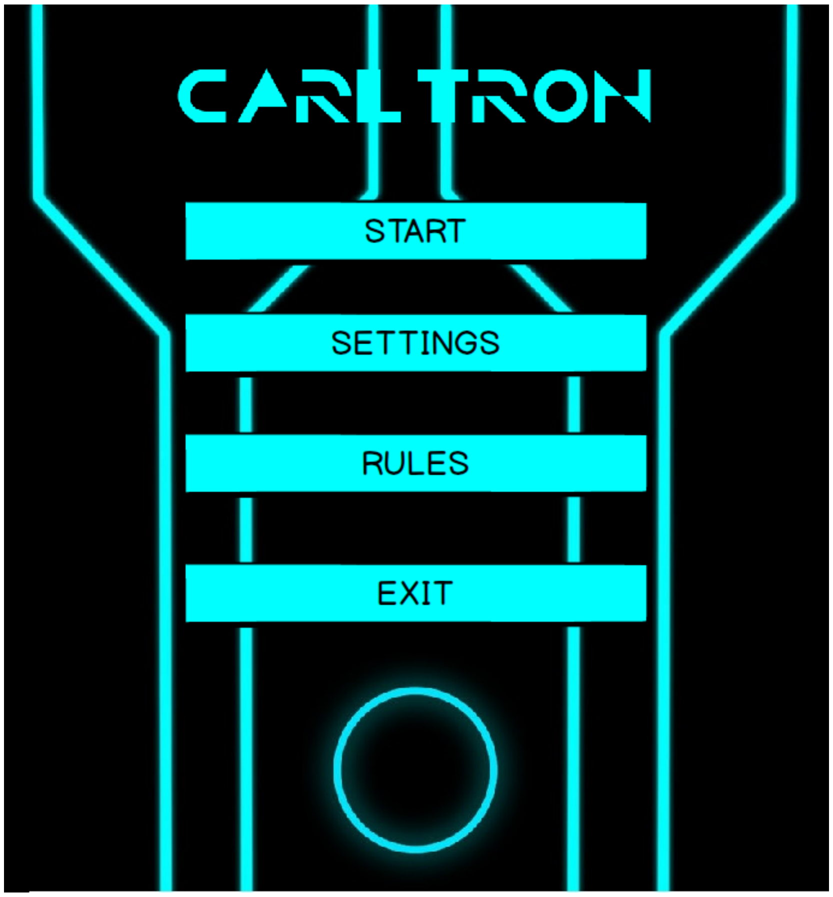
- Grid
- LightCycle

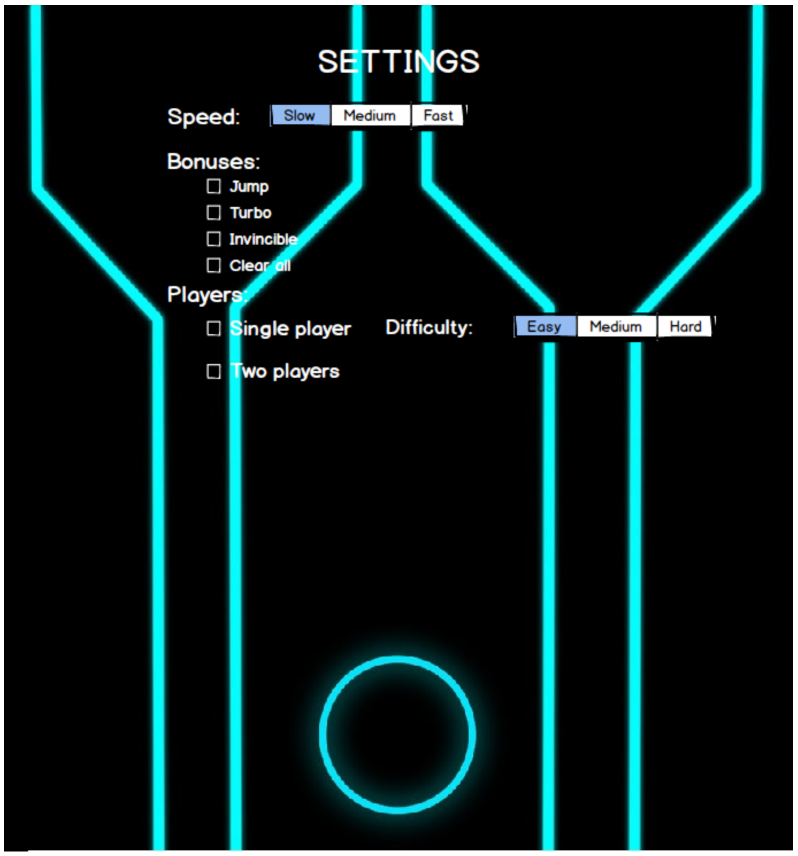
  public void testSetVelocityXTo100ShouldNotAllowTooBigIntegers()
  public void testSetVelocityYTo100ShouldNotAllowTooBigIntegers()
  public void testSetLifeShouldNotAllowDecreaseOfLifeAmountByMoreThan1()
  public void testStepGivesCorrectProgression()
  public void testConsumeShouldUse1TurboAtATime()
  public void testConsumeShouldUse1JumpAtATime()
  public void testConsumeShouldUse1ProtectorAtATime()
  public void testConsumeShouldNotUseATurboWhenEmptyTurbo()
  public void testConsumeShouldNotUseAJumpWhenEmptyJump()
  public void testConsumeShouldNotUseAProtectorWhenEmptyProtector()
- Bonus
- Turbo
- Jump
- Protector

Mockups/storyboards:
Homepage:

contains

- Start: players can start the game by clicking the button
- Settings: players can change general speed, bonuses, mode(single/two players) and the difficulty.
- Rules: players can understand the rules and controls of the game
- Exit: exit the game.





SETTINGS

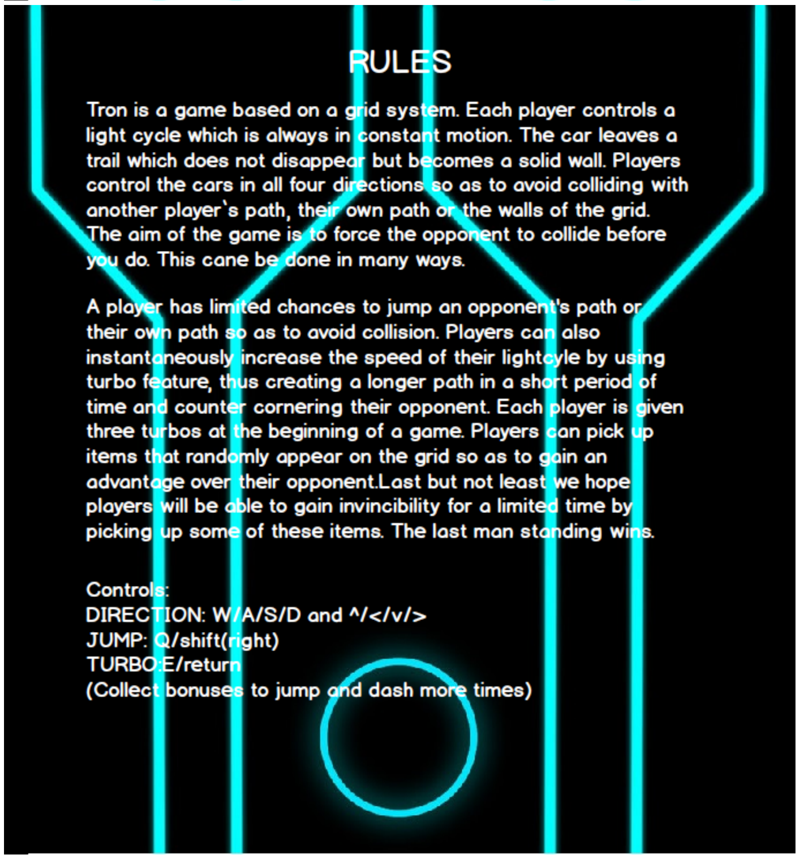
Speed: [✓ Slow] Medium Fast

Bonuses:

- ☐ Jump
- ☐ Turbo
- ☐ Invincible
- ☐ Clear all

Players:

- ☐ Single player   Difficulty: [✓ Easy] Medium Hard
- ☐ Two players

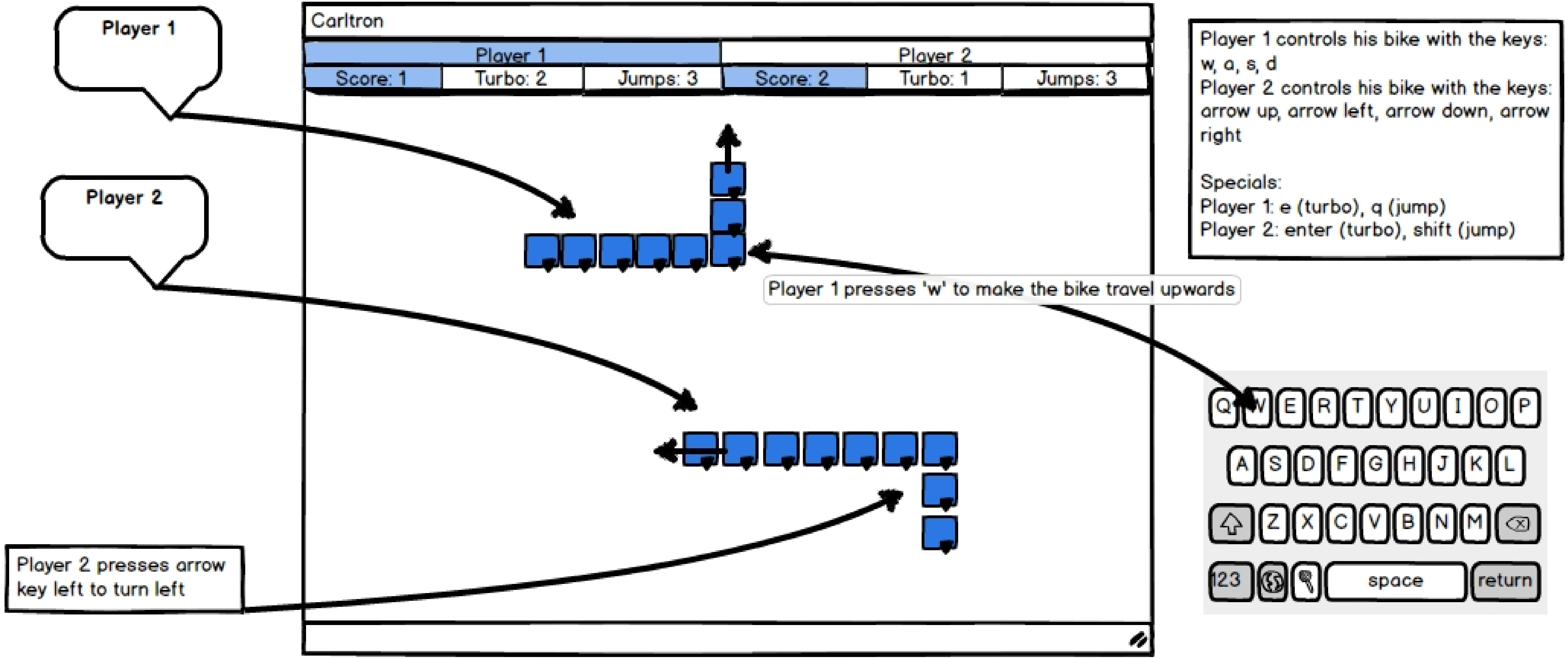


RULES

Tron is a game based on a grid system. Each player controls a light cycle which is always in constant motion. The car leaves a trail which does not disappear but becomes a solid wall. Players control the cars in all four directions so as to avoid colliding with another player`s path, their own path or the walls of the grid. The aim of the game is to force the opponent to collide before you do. This cane be done in many ways.

A player has limited chances to jump an opponent's path or their own path so as to avoid collision. Players can also instantaneously increase the speed of their lightcyle by using turbo feature, thus creating a longer path in a short period of time and counter cornering their opponent. Each player is given three turbos at the beginning of a game. Players can pick up items that randomly appear on the grid so as to gain an advantage over their opponent.Last but not least we hope players will be able to gain invincibility for a limited time by picking up some of these items. The last man standing wins.

Controls:
DIRECTION: W/A/S/D and ^/</v/>
JUMP: Q/shift(right)
TURBO:E/return
(Collect bonuses to jump and dash more times)

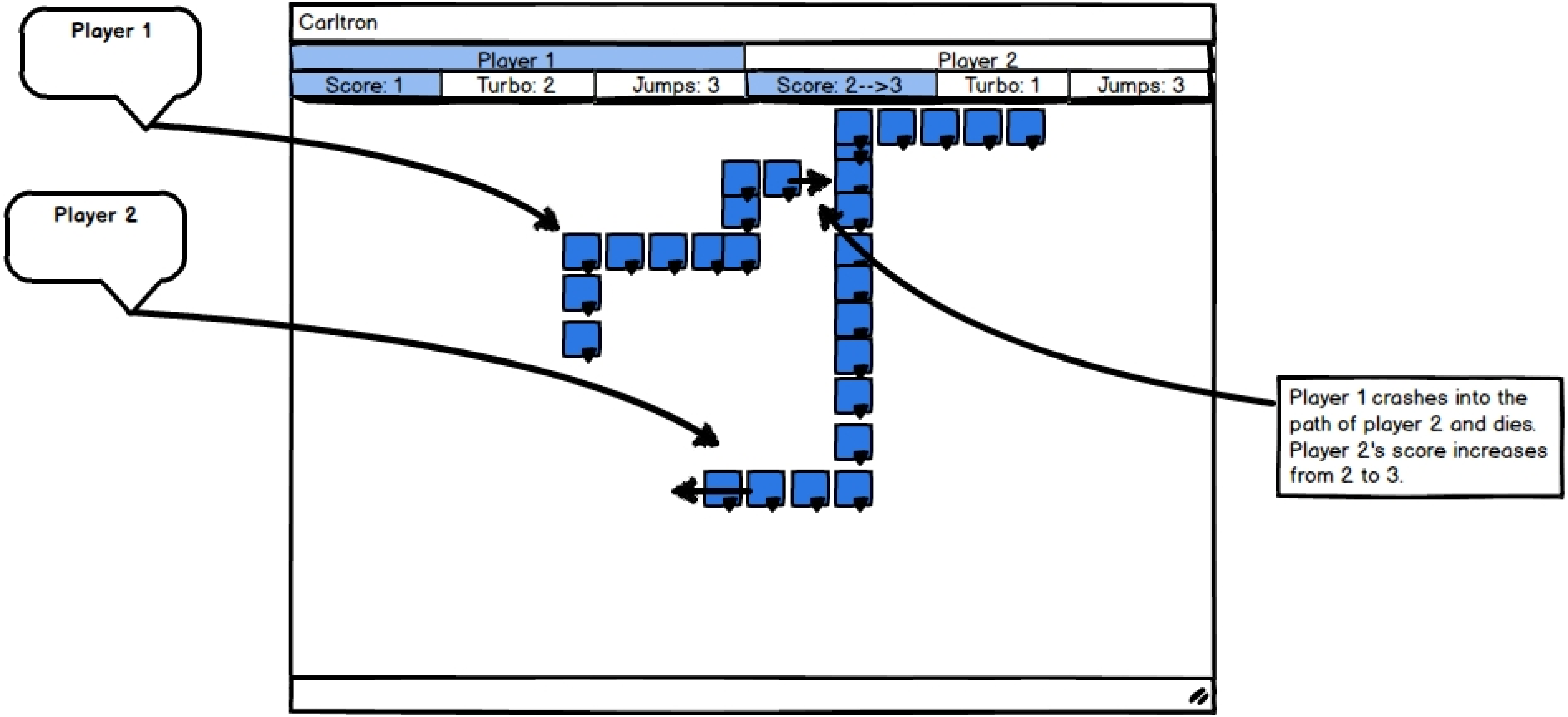


Player 1 presses 'w' to make the bike travel upwards

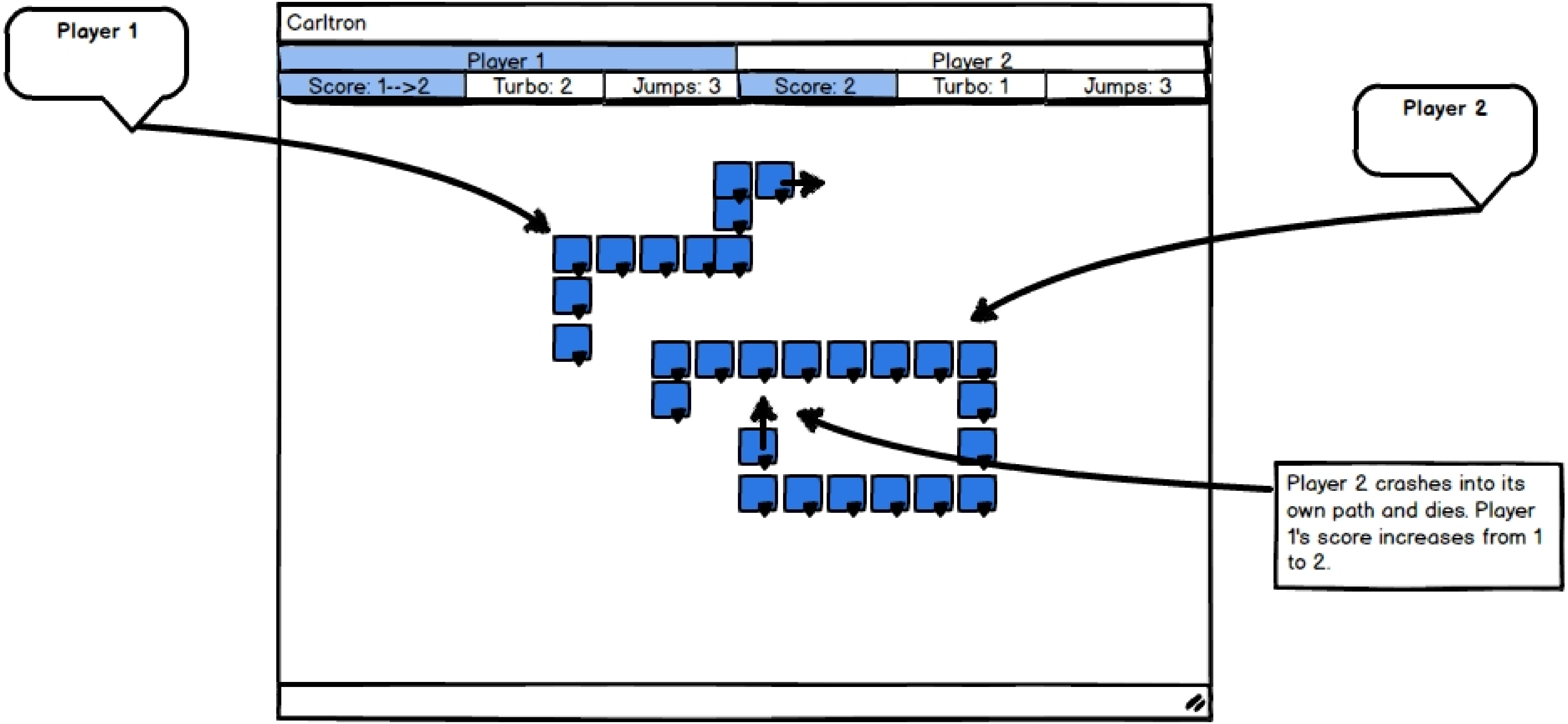
Player 2 presses arrow key left to turn left

Player 1 controls his bike with the keys: w, a, s, d
Player 2 controls his bike with the keys: arrow up, arrow left, arrow down, arrow right

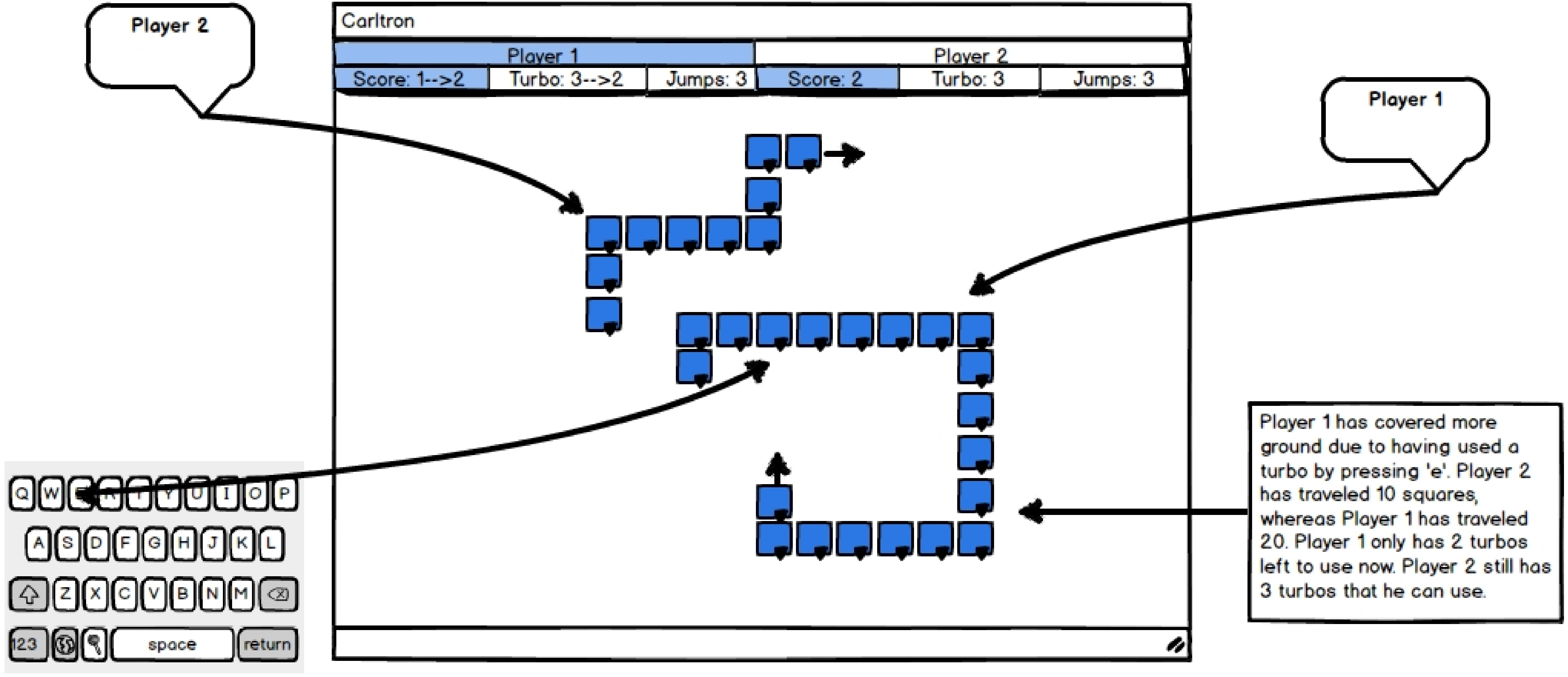
Specials:
Player 1: e (turbo), q (jump)
Player 2: enter (turbo), shift (jump)



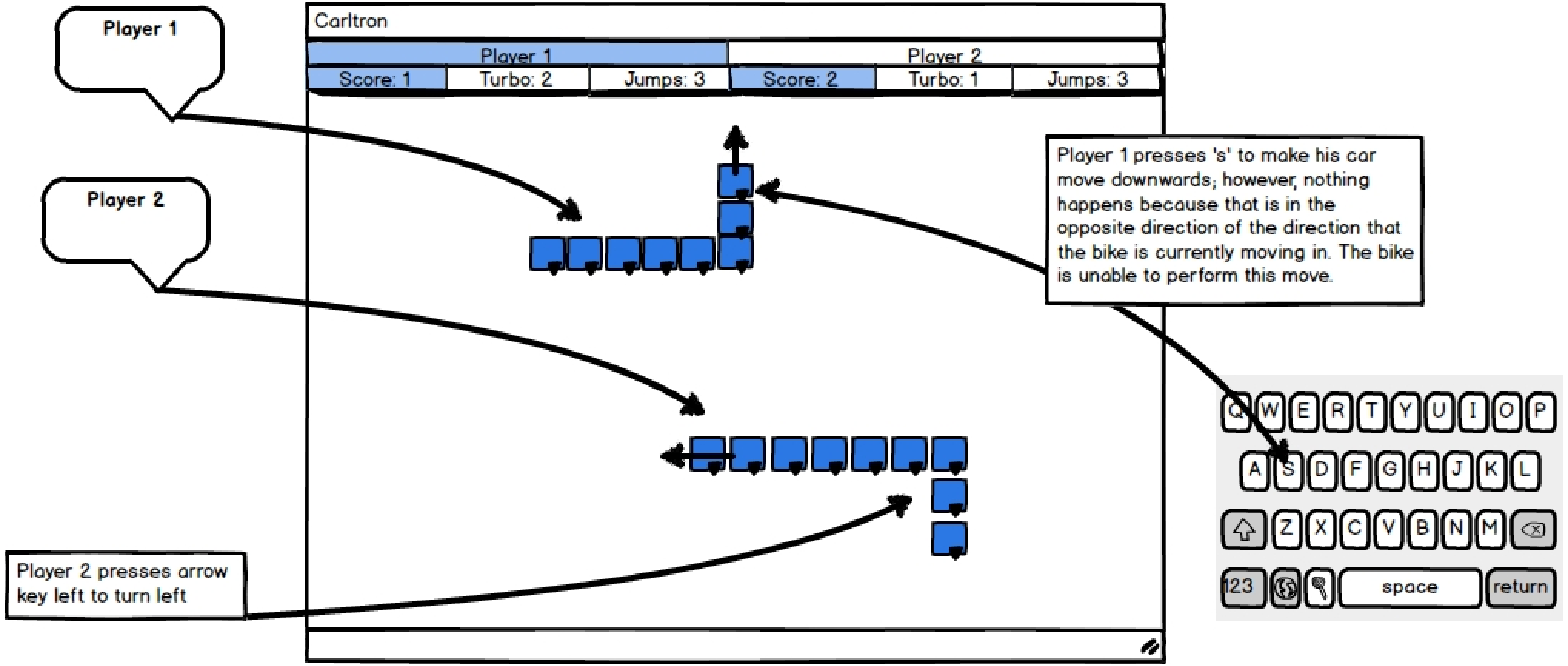
Player 1 crashes into the path of player 2 and dies. Player 2's score increases from 2 to 3.



Player 2 crashes into its own path and dies. Player 1's score increases from 1 to 2.

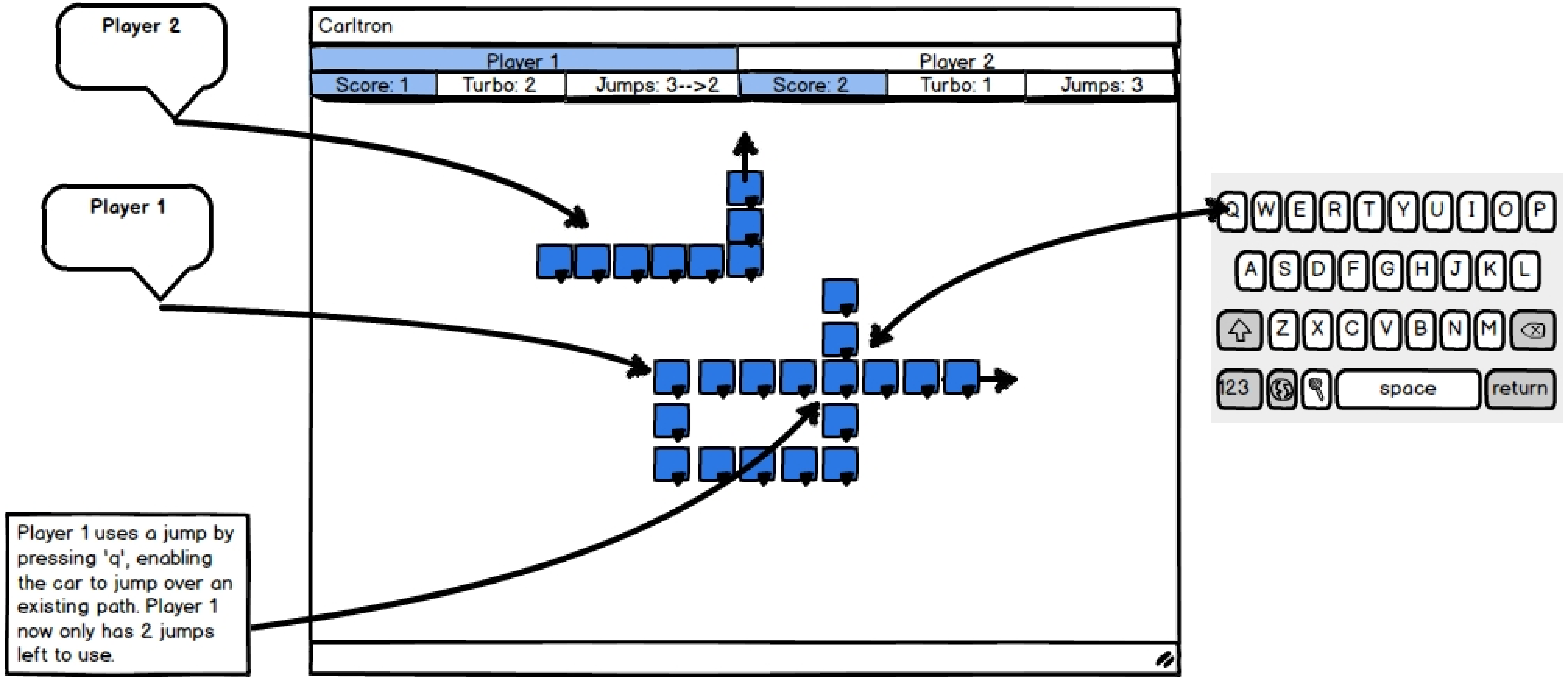


Player 1 has covered more ground due to having used a turbo by pressing 'e'. Player 2 has traveled 10 squares, whereas Player 1 has traveled 20. Player 1 only has 2 turbos left to use now. Player 2 still has 3 turbos that he can use.

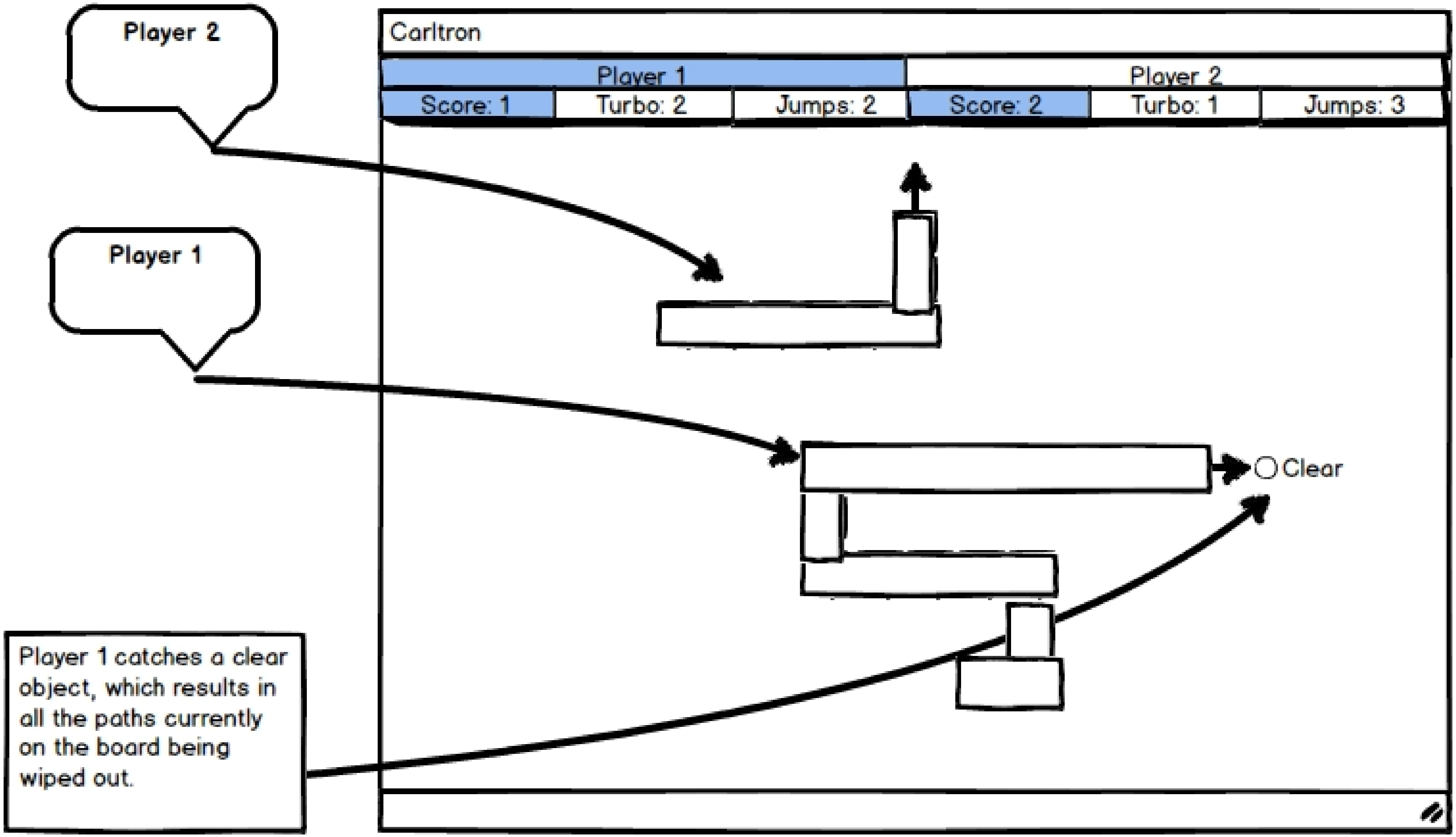


Player 1 presses 's' to make his car move downwards; however, nothing happens because that is in the opposite direction of the direction that the bike is currently moving in. The bike is unable to perform this move.

Player 2 presses arrow key left to turn left

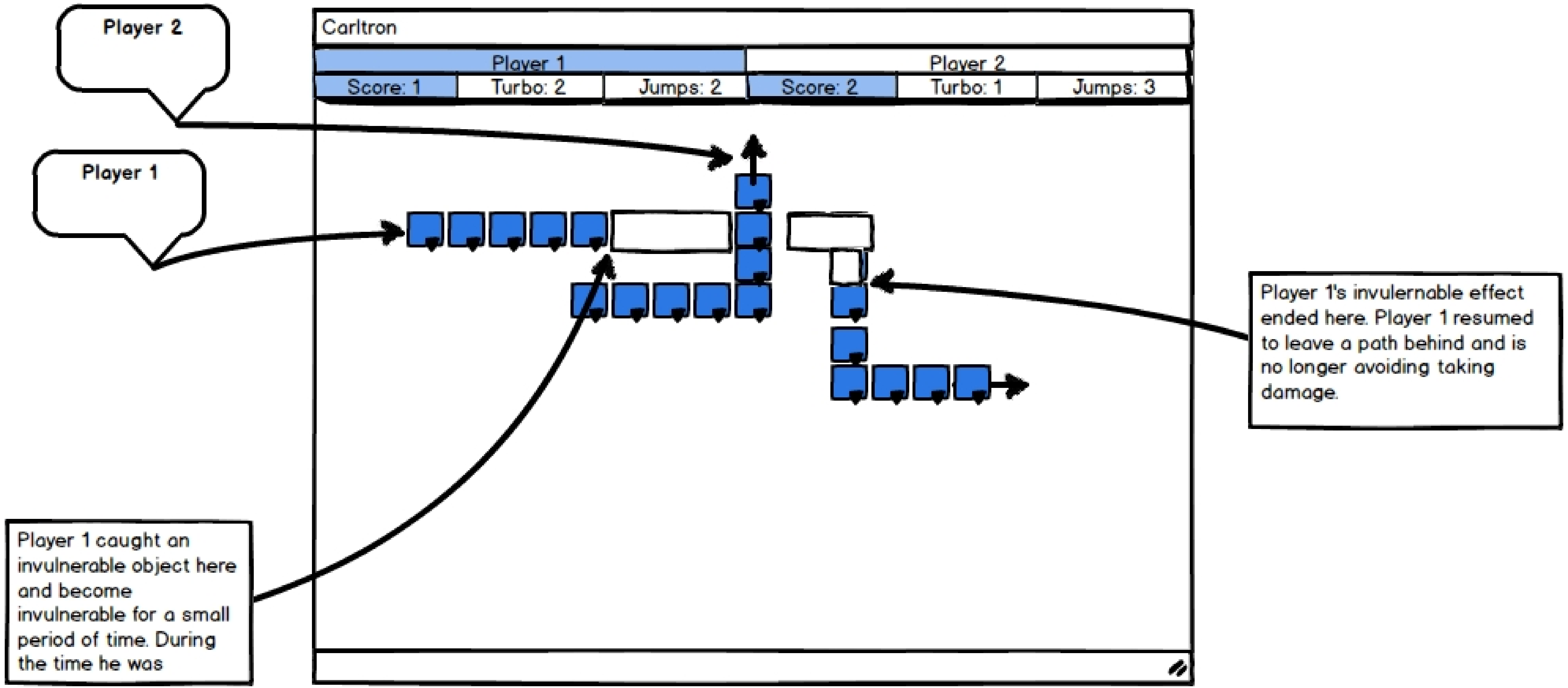


Player 1 uses a jump by pressing 'q', enabling the car to jump over an existing path. Player 1 now only has 2 jumps left to use.



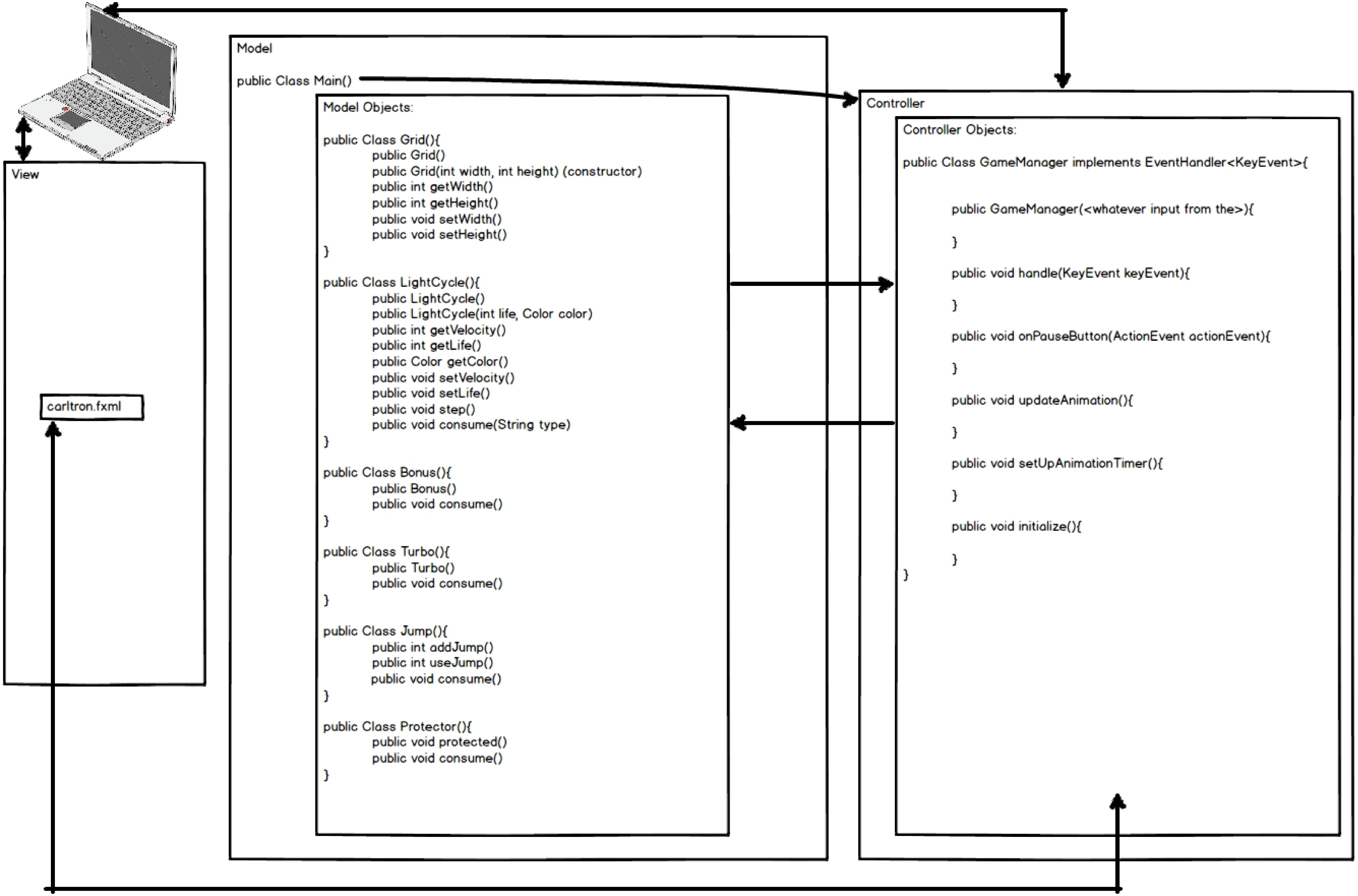
Player 1			Player 2		
Score: 1	Turbo: 2	Jumps: 2	Score: 2	Turbo: 1	Jumps: 3

Player 1 catches a clear object, which results in all the paths currently on the board being wiped out.



Player 1 caught an invulnerable object here and become invulnerable for a small period of time. During the time he was

Player 1's invulernable effect ended here. Player 1 resumed to leave a path behind and is no longer avoiding taking damage.



```
public Class Main()

Model Objects:

public Class Grid(){
        public Grid()
        public Grid(int width, int height) (constructor)
        public int getWidth()
        public int getHeight()
        public void setWidth()
        public void setHeight()
}

public Class LightCycle(){
        public LightCycle()
        public LightCycle(int life, Color color)
        public int getVelocity()
        public int getLife()
        public Color getColor()
        public void setVelocity()
        public void setLife()
        public void step()
        public void consume(String type)
}

public Class Bonus(){
        public Bonus()
        public void consume()
}

public Class Turbo(){
        public Turbo()
        public void consume()
}

public Class Jump(){
        public int addJump()
        public int useJump()
        public void consume()
}

public Class Protector(){
        public void protected()
        public void consume()
}
```

```
Controller Objects:

public Class GameManager implements EventHandler<KeyEvent>{


        public GameManager(<whatever input from the>){

        }

        public void handle(KeyEvent keyEvent){

        }

        public void onPauseButton(ActionEvent actionEvent){

        }

        public void updateAnimation(){

        }

        public void setUpAnimationTimer(){

        }

        public void initialize(){

        }
}
```