Penalty shootout game - Project proposal

This document describes the user experience of a soccer penalty kicks game and its code internal structure, i.e. each class functionalities, attributes and interactions with other classes.

Game description

"A penalty shoot-out (officially kicks from the penalty mark) is a method of determining which team advances or is awarded the championship of an association football match that cannot end in a draw but where the score is tied after the regulation playing time as well as extra time (if used) have expired. In a penalty shoot-out, each team takes turns attempting a specified number of shots on the goal from the penalty mark (5 in FIFA-governed football) that are defended only by the opposing team's goalkeeper, with the team that makes more successful kicks being declared the winner of the match." Source: Wikipedia (https://en.wikipedia.org/wiki/Penalty_shoot-out_(association_football))

Justification

This game is composed of several classes and subclasses that have different types of relationships with each other, either by using other objects attributes as inputs or by having methods that update other classes attributes. I think that the user interface may also be challenging with respect to defining how the inputs will be entered, what the right sound and images will be and the right predictability and complexity levels. All this needs to be considered in order to make the game realistic and fun. For these reasons, I expect the development of this game to be complex enough to meet these project's goals.

User experience

The game will ideally be played by two players, but it can be played by up to 12, grouped in two teams. The user experience of the players involved will follow the process described below.

Configuration

1. Create at least two teams
2. Create at least 6 players per team (one goalkeeper and five shooters)

Game setup

1. Select the two teams that will play the game
2. Select at least 5 players per team

Game:

1. There will be initially 5 pairs of penalty kicks, where the teams will take turns to kick the penalty or catch it.
2. In each kick, if the ball ends up inside the goal, one point will be scored to the kicker's team.
3. If after 5 pairs of kicks the teams end up with the same amount of points, they will move to play one pair of kicks. They will keep playing one pair of kicks until one of the teams misses and the other scores in one of these pairs.

Code structure

Classes

The classes that will compose the game are described below. Details about contents of attributes and inner workings of methods are included when their names are not self-explanatory. The classes are classified into those used to setup the game (Configuration classes), to setup the match (Game setup classes) and to play the game (Game classes).

Configuration classes

Team: represents a team that can be chosen later in a match. Contains data of the players available, the team's track record and has methods to access the players and update its track record.

- Data attributes
  - Name: custom name given to each player
  - Players
  - Wins
- Methods
  - Init: initializes a team instance with its custom name
  - Get player
  - Update wins

Player: base class of shooter and goalkeeper subclasses that contains their identification and abilities information.

- Data attributes
  - Init: initializes a player instance with its custom name
  - Name
  - Ability (randomly assigned)
- Methods
  - Get/set ability

Shooter (subclass of player)": inherits attributes from the Player class, but adds the moves that the kicker is allowed to make.

- Data attributes
  - Name (inherited)
  - Ability (inherited)
  - Moves allowed (class attribute)
- Methods
  - Init: initializes
  - Shoot, with `direction` and `potency` parameters

Goalkeeper (subclass of player): inherits attributes from the Player class, but adds the moves that the goalkeeper is allowed to make.

- Data attributes
  - Name (inherited)
  - Ability (inherited)
  - Moves allowed (class attribute)
- Methods
  - Catch, with `direction` and `potency` parameters

Game setup classes

Match: represents the match in which the shootout will be played. Contains data on the setup of the games and keeps track of its status and results.

- Data attributes
  - Score
  - Kicks: list of kicks associated with this game
  - Number of kicks: tuple that indicates the kick number and the last team that shoot
  - Starting team: identifies the team that started to shoot
  - Teams: tuple with participating teams
  - Score: Keeps score up to last kick
  - Winner: winner of the match, once it's finished, otherwise is None
  - Shooters: list of shooters of each team
- Methods
  - Init: initializes a match instance with the team and players involved
  - Create kick: creates a kick object
  - Get/set teams
  - Get/set shooter
  - Get/set shooter
  - Update score
  - Update winner: the winner will only be updated at the end of the game, but this method will also be in charge of deciding when a team has won.

Game Classes

Kick: setup each kick, with information on each participating player's strategy and the result of the combination of both strategies.

- Data attributes
  - Strategies: tuple containing the strategy of the shooter and the goalkeeper.
  - Result: stores result generated in the result method
- Methods
  - Result: determines kick result from the combination of the values in the `strategies` attribute.

Strategy

- Data attributes
  - Player: player object that decides on the strategy
  - Potency: numeric level of potency with which the ball is shoot
  - Direction: direction given to the kick/catch
  - Margin of error: randomly selected error value within a default range
- Methods
  - Kick destination: calculates the final destination that the ball will have when it reaches the goal

Other features

- The program might have sound effects and images that represent the result of each kick and game. However, this may require the use of libraries outside of the Anaconda distribution.
- Each player's input should be blurred to avoid the other player(s) seeing the user input.
- A random error will be added to each strategy to reduce the predictability of the players's and the adversarie's strategy (e.g. a player can choose a kick in a direction far from the center, which is harder for the goalkeeper to cathcc, but this will imply a higher probability that the ball misses the soccer goal).