Source Code

Create a turn-based combat system

Learn how to create the turn-based combat system found in games like Pokémon, Final Fantasy, and Undertale

AUTHOR
RIK CROSS

In the late 1970s, high school student Richard Garriott made a little game called *Akalabeth*. Programmed in Applesoft BASIC, it helped set the template for the role-playing genre on computers. Even today, turn-based combat is still a common sight in games, with this autumn's *Pokémon Sword and Shield* revolving around a battle system which sees opponents take turns to plan and execute attacks or defensive moves.

The turn-based combat system in this article is text-only, and works by allowing players to choose to defend against or attack their opponent in turn. The battle ends when only one player has some health remaining.

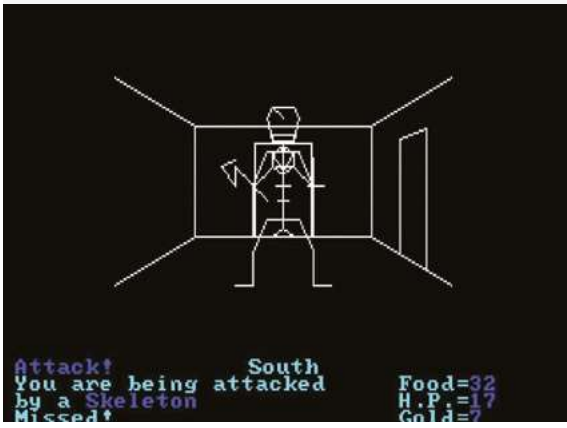
Each `Player` taking part in the battle is added to the static `players` list as it's created. Players have a `name`, a `health` value (initially set to `100`) and a Boolean `defending` value (initially set to `False`) to indicate whether a player is using their shield. Players also have an `inputmethod` attribute, which is the function used for getting player input for making various choices in the game. This function is passed to the object when created, and means that we can have human players that give their input through the keyboard, as well as computer players that make choices (in our case simply by making a random choice between the available options).

With their emphasis on trading and collecting as well as turn-based combat, the *Pokémon* games helped bring RPG concepts to the masses.

A base `Action` class specifies an action `owner` and an `opponent`, as well as an `execute()` method which has no effect on the game. Subclasses of the base class override this `execute()` method to specify the effect the action has on the `owner` and/or the `opponent` of the action. As a basic example, two actions have been created: `Defend`, which sets the owner's `defending` attribute to `True`, and `Attack`, which sets the owner's `defending` attribute to `False`, and lowers the opponent's `health` by a random amount depending on whether or not they are `defending`.

> **"Even today, turn-based combat is still a common sight in games"**

Players take turns to choose a single action to perform in the battle, starting with the human 'Hero' player. The `choose_action()` method is used to decide what to do next (in this case either attack or defend), as well as an opponent if the player has chosen to attack. A player can only be selected as an opponent if they have a `health` value greater than 0, and are therefore still in the game. This `choose_action()` method returns an `Action`, which is then executed using its `execute()` method. A few `time.sleep()` commands have also been thrown in here to ramp up the suspense!



It may look crude, but Richard Garriott's *Akalabeth* laid the groundwork for *Ultima*, and was one of the earliest CRPGs.

After each player has had their turn, a check is done to make sure that at least two players still have a `health` value greater than 0, and therefore that the battle can continue. If so, the static `get_next_player()` method finds the next player still in the game to take their turn in the battle, otherwise, the game ends and the winner is announced.

Our example battle can be easily extended in lots of interesting ways. The AI for choosing an action could also be made more sophisticated, by looking at opponents' `health` or `defending` attributes before choosing an action. You could also give each action a 'cost', and give players a number of action 'points' per turn. Chosen actions would be added to a list, until all of the points have been used. These actions would then be executed one after the other, before moving on to the next player's turn.

Turn-based combat in Python

Here's Rik's code snippet, which creates a simple turn-based combat sequence in Python. To get it running on your system, you'll first need to install Pygame Zero – you can find full instructions at **wfmag.cc/pgzero**


```
import random, time

class Action():
    def __init__(self, owner, opponent):
        self.owner = owner
        self.opponent = opponent

    def execute(self):
        pass

class Attack(Action):
    def __init__(self, owner, opponent):
        super().__init__(owner, opponent)

    def execute(self):
        self.owner.defending = False
        if self.opponent.defending:
            hit = random.randrange(10,20)
        else:
            hit = random.randrange(20,40)
        self.opponent.health -= hit
        print('{} is hit! (-{})'.format(self.opponent.name, hit))

class Defend(Action):
    def __init__(self, owner, opponent):
        super().__init__(owner, opponent)

    def execute(self):
        self.owner.defending = True
        print(self.owner.name, 'is defending!')

class Player():
    players = []

    def __init__(self, name, inputmethod):
        self.name = name
        self.inputmethod = inputmethod
        self.health = 100
        self.defending = False
        self.players.append(self)

    def __str__(self):
        description = "Player: {}\n{}\nHealth = {}\
nDefending = {}\n".format(
            self.name,
            '-' * (8 + len(self.name)),
            self.health,
            self.defending
        )
        return(description)

    @classmethod
    def get_next_player(cls, p):
        # get the next player still in the game
        current_index = cls.players.index(p)
        current_index = (current_index + 1) % len(cls.players)
        while cls.players[current_index].health < 1:
            current_index = (current_index + 1) % len(cls.players)
        return cls.players[current_index]

    def choose_action(self):
        print(self.name, ': [a]ttack or [d]efend?')
        action_choice = self.inputmethod(['a', 'd'])
        if action_choice == 'a':
            print('Choose an opponent')
            # build up a list of possible opponents
            opponent_list = []
            for p in self.players:
                if p != self and p.health > 0:
                    print('[{}] {}'.format(self.players.
index(p), p.name))
                    opponent_list.append(str(self.players.
index(p)))
            # use input to get the opponent of player's action
            opponent = self.players[int(self.inputmethod(opponent_
list))]
            return Attack(self, opponent)
        else:
            return Defend(self, None)

def human_input(choices):
    choice = input()
    while choice not in choices:
        print('Try again!')
        choice = input()
    return choice

def computer_input(choices):
    time.sleep(2)
    choice = random.choice(choices)
    print(choice)
    return choice

# add 2 players to the battle, with their own input method
hero = Player('The Hero', human_input)
enemy = Player('The Enemy', computer_input)

# the hero has the first turn
current_player = Player.players[0]
playing = True

# game loop
while playing:

    # print all players with health remaining
    for p in Player.players:
        if p.health > 0:
            print(p, end='\n\n')

    # current player's action executed
    action = current_player.choose_action()
    time.sleep(2)
    action.execute()

    # continue only if more than 1 player with health remaining
    if len([p for p in Player.players if p.health > 0]) > 1:
        current_player = Player.get_next_player(current_player)
        time.sleep(2)
    else:
        playing = False

for p in Player.players:
    if p.health > 0:
        print('**', p.name, 'wins!')
```