Stephen Oh - ohste

Brandon Withinton - withingb

Ben Lee - leebe2

Mark Huynh - huynhma

1. **Introduction**
   - What is the name of your language? LifeLang
   - What is the language's paradigm? Imperative paradigm
   - (Optional) Is there anything especially unique/interesting about your language that you want to highlight up front?
2. **Design**
   - What *features* does your language include? Be clear about how you satisfied the constraints of the feature menu.

     We will include basic data types through the player constructor, obstacle constructors, and state data type. The data types will consist of:

     i. ObstacleState -> (Position, Lethality)
     ii. HumanState -> (Health, Position, Stamina, Feet)
         1. Position -> (Int)
         2. Heath -> (Int) *Value from 0-100
         3. Stamina -> (Int) *Value from 0-100
         4. Feet -> (Up, Down)

Procedures and functions with arguments will be implemented such that they will be able to manipulate the state of a Human by passing values into functions that will be able to execute a series of steps. For example:

- Jump() -> Will move the human's feet up, move them two units to the right, the down (in order to avoid an obstacle)
- Rest(Int) -> Will recover the stamina of a human by the passed value
- Eat(Int) -> Will recover the health of a human by the passed value
- IsOnObsticle(HumanState, ObstacleState) -> Will check if the human's position is the same as an obstacle
- DamageHuman(HumanState, ObstacleState) -> Will decrease the health of the human by the lethality of the obstacle

We will include conditionals by checking the state of the player against the state of obstacles. For example, if the position of the player matches the position of an obstacle, such as a rock, and their feet are down, then the player will lose health.

We will include variables through the core data types. In the core data types, we will allow for referencing, declaring, and binding as the constructors of the data type. Variables will be mutable with dynamic scope by implementing core data types for re-binding.

We will include recursion/loops by implementing the damage-dealing function that recursively deals damage to the player's health and returns the newly updated health value.

We will include strings and operations by implementing a function to pretty print an array of commands given by the user, as well as a function to concatenate two strings. For the former, when given an array of commands, it will return a string that translates the commands into an easily readable format.

We will include list/array data type and operations by allowing the user to input a list of commands as a type of movement commands for the language to interpret. For example, when the user inputs a list of movements to do, it will concatenate the list and execute it in the order of the list.

- What *level* does each feature fall under (core, sugar, or library), and how did you determine this?
  i. The basic data types fall under the core because they are the backbone for our entire program to run. Without them, we would not be able to express any program.
  ii. The conditions fall under core because we'll implement if-then-else as part of the core data types, specifically under a "statement" data type.
  iii. The loop type structures and recursion states found in our language would become a part of the core because one of the core features of the game is to take damage when the game decides to deal damage to the player.
  iv. The variable/local names fall under core because we will implement all of these features through the core data types.
  v. The "while" loop would fall under core because we'll also implement this as one of the core data types.
  vi. The strings and operations fall under the library because it will include functions, such as concatenating two strings or pretty printing, that can be imported.
  vii. The list/array data type and operations fall under the library because it will include functions, such as appending a value to an array or loop through an array, that can be imported.
- What are the *safety properties* of your language? If you implemented a static type system, describe it here. Otherwise, describe what kinds of errors can occur in your language and how you handle them.

  We will be implementing a static type system. Types are static as we declare them. We do not need to convert between the variable types to decipher values throughout the usage of the language. We will implement checks to catch type errors prior to running the program.

3. **Implementation**
   - What *semantic domains* did you choose for your language? How did you decide on these?
     i. State -> State type semantic domain. The initial state will be the health and position of the current character. After some action, the character will have another health and position.
   - Are there any unique/interesting aspects of your implementation you'd like to describe? It draws inspiration from the pen program and car program examples. We, however, add additional features that will hopefully entertain the user.