PLT Project Proposal

Team members:
Albert Cui (aqc2109): Language Guru
Karen Nan (kkn2109): Project Manager
Michael Raimi (mar2260): System Architect
Mei-Vern Then (mt2837): Verification and Validation

Purpose and Goals

The GAWK language will facilitate developers who want to create RPG (role-playing games). In RPGs and decision-based games, the user is often given two or three choices and asked to make a decision, thereby creating an individual storyline unique to user.

GAWK simplifies the process of creating and designing a decision-based game by providing a template based on directed graphs, nodes, and weighted edges. We propose a design in which nodes represent each "world" or location in the game, which when connected together becomes a web of environments with explicit relationships between the "player" and to other "worlds."

The power of this language comes in the idea of predicates, which simplifies the process of applying changes to the environments, or "worlds" of the game. As the current "world" or node is updated, the language evaluates predicates attached to the worlds to update the state of the game. Each predicate has a block attached to them that are executed if the predicate is evaluated as true. We hope to provide a robust randomize function call that integrates with our predicate system enabling the possibility of a truly dynamic world. This allows for change and variation in the current state of the player as different functions may be called.

As creators of the GAWK language, we seek to develop a method to produce these games in an intuitive manner so that RPG developers can focus more time on the design of these games and less on the technical challenges associated with the process.

Language Data Structures

Primitives

- Boolean: variables that hold only logical true or false values
- Character: unit of information to hold letters of the English alphabet or numbers
- Integers: units to hold whole numbers
- Floating-point numbers: units to hold partial numbers that allow for decimal representations of numbers
- Null: represents an uninitialized, undefined, empty, or meaningless value
- Node: "world"
    - Contains user defined properties that represent the program's current state
    - Contains:
        - Predicates
        - Neighbors, with travel costs for each neighbor node, specified by the load method.

Data Structures

- Predicate
    - Boolean expression that is evaluated for truthfulness
    - Predicates have blocks attached to them that are executed if the predicate is evaluated as true
    - Employs integrated randomization engine that can aid in determining dynamic events with the "case" keyword

- Structs
    - User defined data structures.
    - Do not contain methods/functions.
    - No constructor

Operators

Symbol	Function	Example
+ - * / ++ --	Addition: add operands on either side of the operator Subtraction: subtract right hand operand from left hand operand Multiplication: multiplies operands on either side of the operator Division: divide left hand side operand by right hand side operand Incremental addition: adds value of 1 to left hand side operator Incremental subtraction: subtracts value of 1 to left hand side operator	4+6 gives 10 5-2 gives 3 8*7 gives 56 9/3 gives 3 5++ gives 6 5-- gives 4
== != > < >= <=	Checks if values on either side of operator are equal, if is equal then true Checks if values on either side of operator are unequal, if unequal then true Checks if left operand is strictly greater than right, if greater than then true Checks if left operand is strictly less than right, if less than then true Checks if left operand is greater than or equal to right, if yes then true Checks if left operand is less than or equal to right, if yes then true	(a==b) gives false (a!=b) gives true (2<3) gives true (2>3) gives false (5>=2) gives true (5<=2) gives false
= += -=	Assignment: sets left hand side operand to value of right hand side operand Addition assignment: adds left hand side operand to right hand side operand and assigns result to the left hand side operand Subtraction assignment: subtracts right hand side operand from left hand side operand and assigns result it to the left hand side operand	a = b + c a += b is a = a + b a -= b is a = a- b
& |	Logical AND operator, binary Logical OR operator, binary	
// /* */	Comments to explain code for user purposes, not compiled as actual code Comments for blocks of texts that span multiple lines	//This is a comment /* This is also a comment */
.	Accesses fields of a struct	p.hp accesses the hp field of p

Test Program

```
// This is an example program for a text-based RPG

struct Playah { //this defines a struct. It has two fields.
        int hp;
        int gold;
};

// This specifies global predicates that are evaluated at each node,
// before each node's instructions block
// These are evaluated at run time. If p does not exist, or does not
// have field hp, an exception will be thrown
global {
        (p.hp <= 0 ) {
                exit("You died! Game over"); //Terminates a program,
                                                        // printing the string.
        }
        (p.gold > 100) {
                exit("You win!");
        }
}

Node n1 {
        // Represents state in this node.
        state {
                Playah p = new Playah();
                p.hp = 10;
                p.gold = 15;
        }

        // The instructions block executes before local predicates are evaluated
        instructions {
                print(
                        "You reach the end of the hall. There is a door to your left and a door to your right. 1. Open the
left door 2. Open the right door."
                );
        }

        // These are predicates specific to this node. They are evaluated top to bottom.
        // Predicates as strings automatically evaluate against stdin.
        // This will thus block until input is received.

        ("1") {
                        //this predicate determines which case to call based
                        //on an implicit probability engine. The arguments //formulate a threshold, determining the
                        probability //of each case.
```

```
            case(p.skill, 10){
                print(
                    "You fall through a trap door! HP -10"
                );
            } case(){  //this case is empty because it implicitly uses
                                        // the same args as in the first case
                print(
                    "You avoided a trap door! HP +5"
                );

            }
            load(n2, p.hp(-10)); //This predicate loads a neighbor node
        }
        ("2") {  //if no cases are listed all lines are executed
            print(
                "You find treasure! gold +100"
            );
            load(n2, p.gold(100));
        }

        (default) { // This will always be evaluated, provided a load call wasn't made.
            print(
                "That wasn't an option! Try again."
            );
            reset; //This causes predicate evaluation to begin at the
                //top again.
        }
}

Node n2 { // state automatically inherits from previous nodes.
        // This doesn't happen since the game will terminate from
        // global predicate evaluation
        before {
            print(
                "There's a yeti! 1. Attack with sword. 2. Attack with shield"
            );
        }

        ("1") {
            print(
                "That wasn't a good idea… You miss and get destroyed."
            );
        }

        ("2") {
            print(
                "That wasn't a good idea… You miss and get destroyed."
```

```
                );
        }

        (default) { // This will always be evaluated, provided a load call wasn't made.
                print(
                        "That wasn't an option! Try again."
                );
                reset; //This causes predicate evaluation to begin at the
                        //top again.
        }
}

start(n1); //This starts program execution at n1.
```

// Example program output

>> You reach the end of the hall. There is a door to your left and a door to your right. 1. Open the left door 2. Open the right door.
>> 1
>> You fall through a trap door! HP -10
>> You died! Game over

// Another run

>> You reach the end of the hall. There is a door to your left and a door to your right. 1. Open the left door 2. Open the right door.
>> 2
>> You find treasure! gold +100
>> You win!