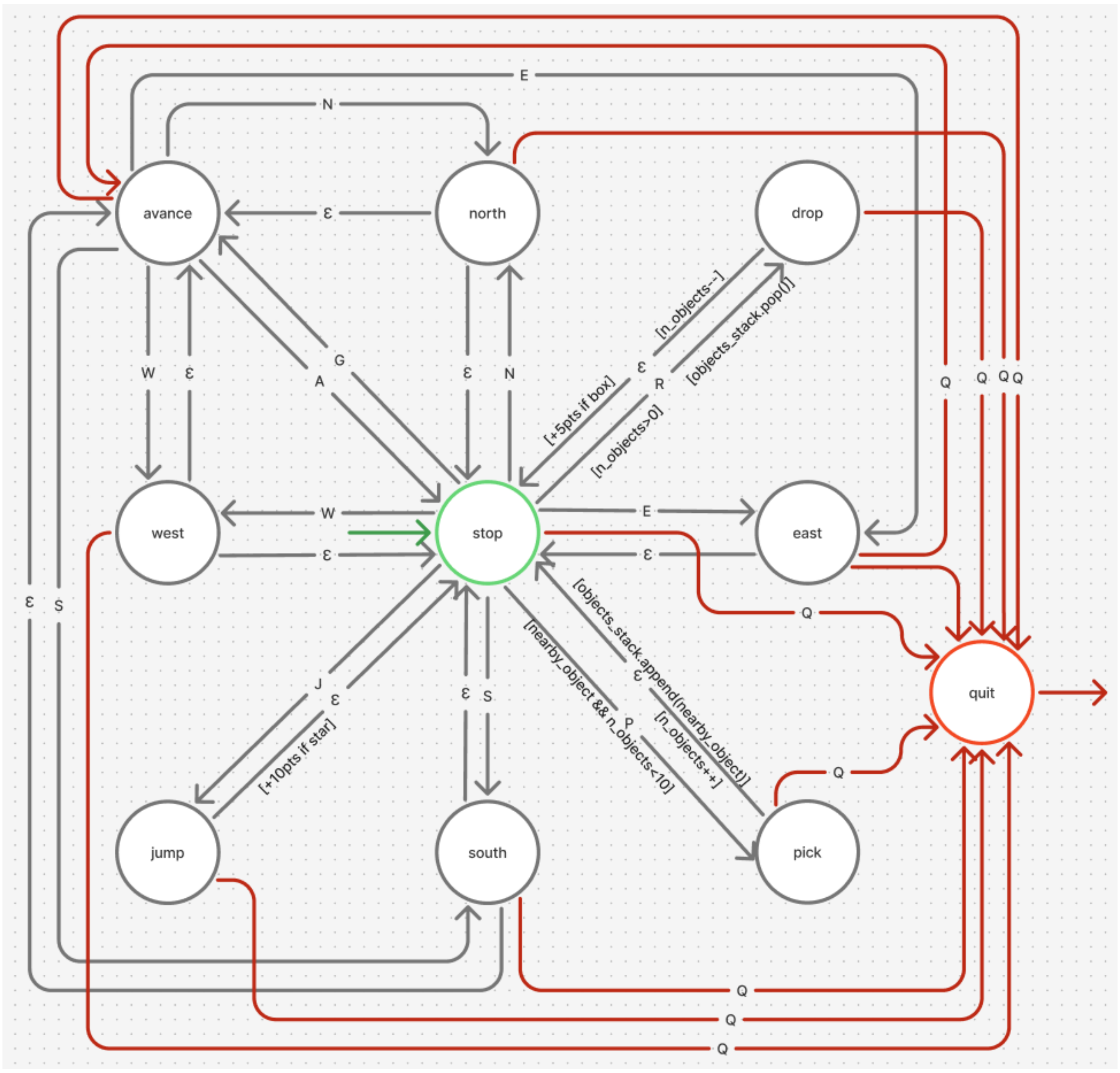
TP 1 - Game Character FSM

Khalil MZOUGHI 2A FISA INFO

1. Graphical representation of the FSM

The finite state machine used in this assignment is based on the following diagram:



This diagram shows a complete model of the game character's behavior with the following structure:

Variables:

- `n_objects: int`: the count of objects hold by the player
- `nearby_object: boolean`: `true` if there is a nearby object, else, `false`
- `star: boolean`: `true` if the jamp the jump allows the character to pass over a **star**, else, `false`

- `box: boolean`: `true` if the drop-off is in the **box**, else, `false`

States:

- `stop`: Central state where the character is stationary (initial state, highlighted in green)
- `north`, `east`, `south`, `west`: Orientation states
- `avance`: Movement state - character is advancing in current direction
- `jump`: Character jumps
- `pick`: Character picks up an object
- `drop`: Character is drops an object
- `quit`: Terminal state ending the simulation (highlighted in red)

Key transitions:

- Direction keys ('N', 'E', 'S', 'W') change the character's orientation
- 'G' starts movement from `stop` to `avance`
- 'A' stops movement from `avance` to `stop`
- 'J' initiates jump from `stop` to `jump`
- 'P' with condition `[nearby_object && n_objects<10]` allows object pickup
- 'R' with condition `[n_objects>0]` allows object dropping
- 'Q' from any state leads to `quit`
- ε-transitions are used for automatic returns to states after actions

Features:

- Stack to track objects collected (`n_objects`) with maximum of 10
    - `[objects_stack.append(nearby_object)]` adds the nearby object ti the stack of objects hold by the character
    - `[objects_stack.pop()]` pops the first hold object in the player stack
- Point awards (`[+10pts if star]`, `[+5pts if box]`)
- Conditional transitions based on game state

Design choices:

- The FSM separates orientation (north, east, south, west) from movement state (avance) to accurately track the character's direction.
- Epsilon transitions model automatic state changes after actions like jumping or picking up objects.
- Conditions enforce game rules such as object limits and prerequisites for actions.
- All states have a direct path to the quit state via the 'Q' transition.
- Color coding is used to enhance readability:
    - Green circle for the initial state (`stop`)
    - Red paths for all exit transitions to `quit`

2. Grammar for FSM text file

The following formal grammar describes the syntax used for storing the FSM in a text file:

```
<FSM_File>              → <Header> <States_Section> <Transitions_Section> <Footer>

<Header>                → "FSM" L <FSM_Name> L "BEGIN" L

<States_Section>        → "STATES" L <States_List> "END-STATES" L
<States_List>           → <State> L <States_List> | <State> L
<State>                 → <StateName> <StateType>
<StateType>             → ",INITIAL" | ",FINAL" | ",INITIAL,FINAL" | ε
<StateName>             → <lowercase_letter> <state_name_tail>
<state_name_tail>       → <lowercase_letter> <state_name_tail> |
<lowercase_letter> | ε
<lowercase_letter>      → "a" | "b" | ... | "z"

<Transitions_Section>   → "TRANSITIONS" L <Transitions_List> "END-TRANSITIONS" L
<Transitions_List>      → <Transition> L <Transitions_List> | <Transition> L
<Transition>            → <SourceState> "->" <InputSymbol> <Condition> "->"
<TargetState> <Action>
<SourceState>           → <StateName>
<InputSymbol>           → <Symbol> | "epsilon"
<Symbol>                → <letter> | <digit>
<Condition>             → "[" <ConditionExpr> "]" | ε
<ConditionExpr>         → <SimpleCondition> | <SimpleCondition> <LogicalOp>
<ConditionExpr>
<SimpleCondition>       → <Variable> <ComparisonOp> <Value>
<TargetState>           → <StateName>
<Action>                → "[" <ActionExpr> "]" | ε
<ActionExpr>            → <SimpleAction> | <SimpleAction> "," <ActionExpr>
<SimpleAction>          → <Variable> <AssignmentOp> <Value> | <MessageAction>

<Footer>                → "END" L

<Variable>              → <lowercase_letter> <identifier_tail>
<identifier_tail>       → <lowercase_letter> <identifier_tail> | <digit>
<identifier_tail> | "_" <identifier_tail> | ε
<Value>                 → <Number> | <Boolean> | <String>
<Number>                → <digit> <Number> | <digit>
<digit>                 → "0" | "1" | ... | "9"
<Boolean>               → "true" | "false"
<String>                → """" <character_sequence> """"
<ComparisonOp>          → "==" | "!=" | "<" | ">" | "<=" | ">="
<LogicalOp>             → "&&" | "||"
<AssignmentOp>          → "++" | "--" | "+=" | "-=" | "="
<MessageAction>         → <String>
```

Where:

- L represents a line break
- ε represents an empty string

3. FSM Back up file :

Here is the complete text file representing the FSM according to the grammar defined above:

```
FSM
game_character
BEGIN

STATES
stop,INITIAL
north
east
south
west
avance
jump
pick
drop
quit,FINAL
END-STATES

TRANSITIONS
stop -> G -> avance
avance -> A -> stop
stop -> N -> north
stop -> E -> east
stop -> S -> south
stop -> W -> west
north -> epsilon -> avance
east -> epsilon -> avance
south -> epsilon -> avance
west -> epsilon -> avance
avance -> N -> north
avance -> E -> east
avance -> S -> south
avance -> W -> west
north -> N -> north
east -> E -> east
south -> S -> south
west -> W -> west
stop -> J -> jump
jump -> epsilon -> stop [points+=10 if star]
stop -> P [nearby_object && n_objects<10] -> pick
pick -> epsilon -> stop [n_objects++] [objects_stack.append(nearby_object)]
stop -> R [n_objects>0] -> drop
drop -> epsilon -> stop [n_objects--, points+=5 if box] [objects_stack.pop()]

stop -> Q -> quit
avance -> Q -> quit
north -> Q -> quit
east -> Q -> quit
south -> Q -> quit
west -> Q -> quit
jump -> Q -> quit
pick -> Q -> quit
```

```
drop -> Q -> quit
END-TRANSITIONS

END
```

Implementation details:

1. **Header section**: Identifies the file as an FSM description with a name "game_character".
2. **States section**: Lists all 10 states in the FSM:
   - `stop` is marked as the initial state
   - `quit` is marked as the final state
   - All other states are regular states without special properties
3. **Transitions section**: Defines all possible state changes:
   - Movement controls: 'G' to start moving, 'A' to stop
   - Direction controls: 'N', 'E', 'S', 'W' to change orientation
   - Action transitions: 'J' for jump, 'P' for pick up, 'R' for drop
   - Automatic (epsilon) transitions after actions
   - 'Q' transitions from all states to quit the simulation
4. **Conditions and Actions**:
   - `[nearby_object && n_objects<10]`: Ensures objects can only be picked up when in range and under the limit
   - `[n_objects>0]`: Ensures objects can only be dropped when the character has at least one
   - `[points+=10 if star]`: Adds points when jumping through stars
   - `[points+=5 if box]`: Adds points when dropping objects in the box
   - `[n_objects++]`: Increments the object counter after picking up
   - `[n_objects--]`: Decrements the object counter after dropping
   - `[objects_stack.append(nearby_object)]` adds the nearby object ti the stack of objects hold by the character
   - `[objects_stack.pop()]` pops the first hold object in the player stack