Game Generator Document v1.0

VERSIONING

Version	Date	Author	Description
V 1.0	19-Nov-2015	Allen Zhong	Initial release

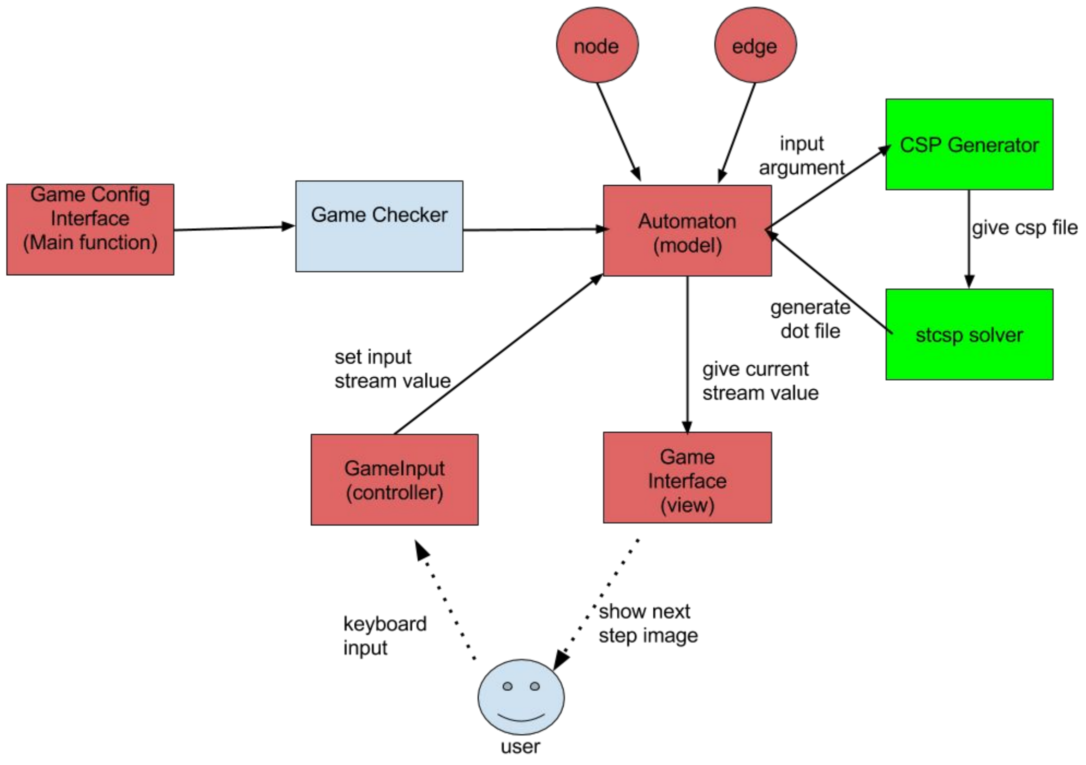
TABLE OF CONTENTS

1.Project Description

Game Generator is a java project which can generate “game and watch” automatically by simple configuration and sprite preparation. The game engine depends on the Stream Constraint Satisfaction Problem solver. For more details please refer to *Constraint Programming on Infinite Data Stream*.

Game Generator contains three major components, including game configurator, csp generator and game engine. These three components work sequentially. Game configurator is to show configuration, parse the text and pass them to other components. Csp generator is to generate the csp file and run the stream constraint satisfaction problem solver to get the dot file as the game automaton. Game engine is a model-view-controller structure to run the game and watch.

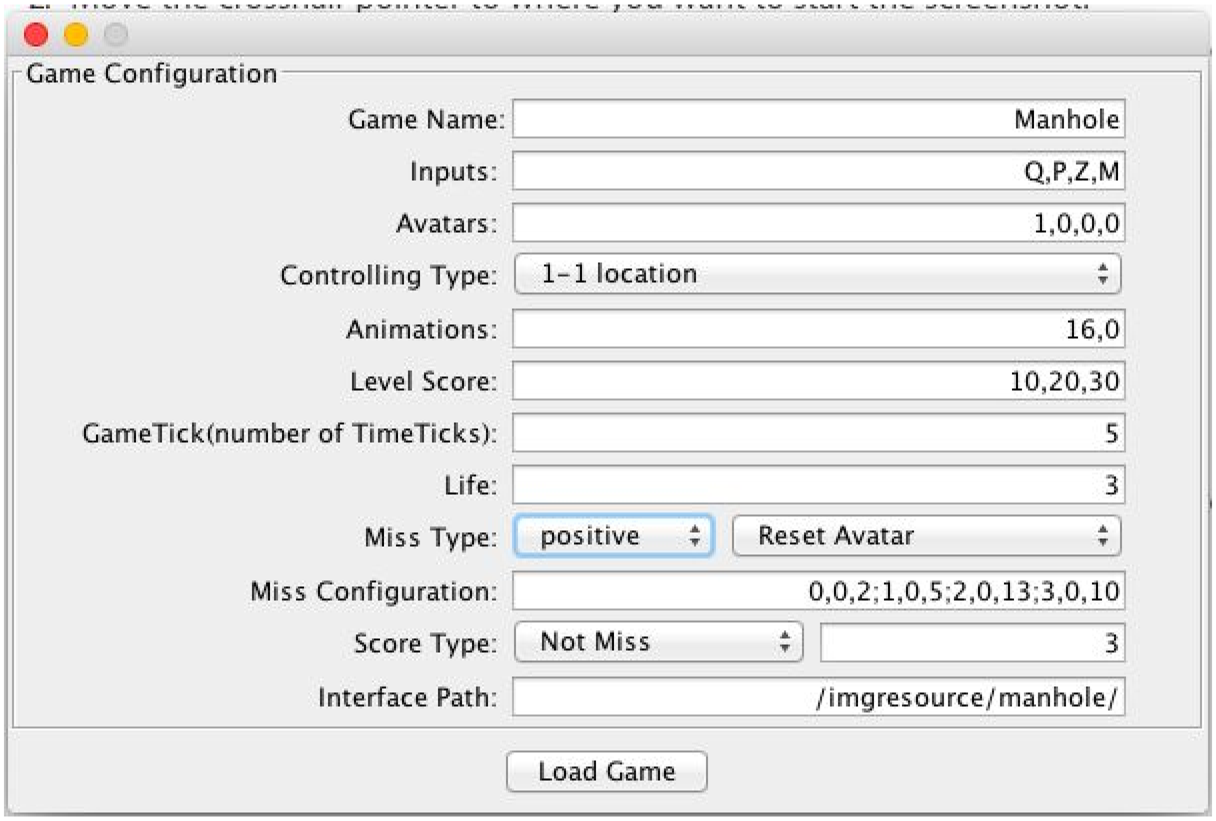
2.System Diagram



2.2. Component Description

2.1. Game configurator

The game configurator would shows the configuration window and parse the text entered by users. A sample window is shown below:



Game Configuration

Game Name:	Manhole
Inputs:	Q,P,Z,M
Avatars:	1,0,0,0
Controlling Type:	1-1 location
Animations:	16,0
Level Score:	10,20,30
GameTick(number of TimeTicks):	5
Life:	3
Miss Type:	positive / Reset Avatar
Miss Configuration:	0,0,2;1,0,5;2,0,13;3,0,10
Score Type:	Not Miss / 3
Interface Path:	/imgresource/manhole/

The game configuration is described as followed:

- GameName: the given name of the game and csp file
- Inputs: inputs for the game, should be an array of character separated by comma
- Avatars: the states of avatar locations, should be an array of “0” and “1” which indicates the initial states of avatar locations. Only one “1” in the array is accepted.
- Controlling Type: the control type of the game, it can be either “1-1 location” or “1-1 step”. “1-1 location” means that different inputs would set the avatar in different location, like control in “Manhole” game. “1-1 step” means that inputs would set the avatar to the neighbour location, like control in “Octopus” game.
- Animations: animation setting takes a string separated by comma and semicolon. The semicolons separate different tuple of animation and colons separates different attribute(step and trace) of animation.

- Level Score: an array of score separated by comma for different levels. For now it is not used in game engine.
- GameTick: the number of timetick for each game tick. In each game tick, the animation and game state would change once. In each time tick, the avatar can change location once.
- Life: the number of life in the game. If life equals to zero, the game would be over.
- Miss Type: the condition to trigger miss and the miss resset. The condition can be either “positive” or “negative”. “Positive” means that if both some states of animation is presented and avatar is in some locations, then miss is trigger. “Negative” means that if some states of presented and avatar is not in some locations, then miss is trigger. The miss reset can be either “Reset Animation” or “Reset Avatar”. They mean miss state would reset either some animation states or avatar loaction to zero. For example, “Octopus” is a game with “positive” and “Reset Avatar” setting, and “Manhole” is a game with “negative” and “Reset Animation” setting.
- Miss Configuration: an array of miss configuration tuple separated by semicolon. Each tuple contains avatar location, animation number and animation step, which means if the avatar is (not) in certain location and certain animation is in certain steps, then miss is trigger.
- Score Type: it can either be “Not Miss” or “Avatar Location”. “Not Miss” means that user scores when certain animation is in certain step but miss is not trigger. “Avatar Location” means that user scores when the avatar is in certain location. For example, “Octopus” is a game with “Avatar Location” setting, and “Manhole” is a game with “Not Miss” setting.
- Interface Path: the image folder location. The folder should be in the same location as the game generator.

2.2. Csp generator

The csp generator would generates the csp file that can be parsed by stream constraint satisfaction solver. The corresponding stream csp logic is presented in section 3.

2.3. Game engine

The game engine includes the automaton (model), game interface (view) and game input (controller).
The automaton class mainly manipulate the game state. The constructor would interpret the dot file and store the nodes and edges using Node and Edge class. For each time tick, it would search game states for next step according to keyboard input. The variables in the automaton would exported by the get method.

The game interface class mainly decide whether certain sprites would be shown in the window for each step according to the variables in the automaton. The constructor would load images in the “interface path” set by game configurator. When the game is over, the game interface would stop update the icons.
The game input class mainly listen to keyboard input from user and pass it to the automaton. Besides, it would also maintain the timer so that it would trigger the automaton to search next steps for each time tick.

3.Stream CSP Logic

The variable in csp file is shown as followed:

Variable Name	Description	Domain	First value
Input	Indicates input number by user. zero for no input from user.	[0 , number of user input]	0
Life	Indicates the number of left lives	[0 , number of life]	Life number set by user
Score	Indicates whether user scores or not in each game state	[0 , 1]	0
Gametick	Indicates the game tick of each game state	[0 , number of gametick - 1]	number of game tick
GameOver	Indicates whether game is over (life number is zero) or not)	[0 , 1]	0
Level	Indicates the level in each game state (now is not used)	[0 , number of level -1]	0
Random	Random number for animation display	[0 , 1]	Not set
Avatar	Indicates the location of avatar	[0 , number of avatar location -1]	0
Ava[x]	Indicates whether the corresponding avatar image should be shown, used by Game Interface class	[0 , 1]	Not set
Ani[x]	Indicates the step number for the x^{th} animation	“Trace” : [0 , number of animation*2 -1] “Not Trace”: [0 , number of animation]	0
Ani[x]Seq[y]	Indicates whether the corresponding image for animation x step y should be shown, used by Game Interface class	[0 , 1]	Not set

Miss	Indicates whether miss is trigger in certain game state	[0 , 1]	0
MissLoc[x]	Indicates which location of miss is trigger	[0 , 1]	0

3.1. General Game constraint

- Game over constraint : game is over when life equals to zero.
  <u>GameOver == Life eq 0;</u>

- Gametick constraint : game tick is a periodic sequence of timetick determined by level variable.
  <u>next Gametick == (Gametick+1)%([game tick number] - Level);</u>

- Life constraint : Life number should be greater than zero; when miss is trigger, then life number would decrease by 1.
  <u>next Life == if Life eq 0 then 0 else if (Miss eq 1) and (Gametick eq 0) then (Life-1) else Life;</u>

- Level constraint : For now, level is always zero.
  <u>Level == 0;</u>

3.2. Avatar Location constraint

- Ava[x] variable: Ava[x] variable should be true when Avatar variable is [x]
  <u>Ava[x] == Avatar eq [x];</u>

- Avatar variable: ([x] indicates the number of avatar location)
  - Avatar should not change when game is over or there is no input
    <u>if GameOver or Input eq 0 then Avatar else ...</u>
  - If miss state reset avatar location, then it should be reset when miss is triggered
    <u>if Miss eq 1 then 0 else ...</u>
  - for “1-1 location” control, the avatar value should corresponds to input variable
    <u>(Input-1)</u>
  - for “1-1 step” control, the avatar value should be modified according to input variable
    <u>if Input eq 1 and Avatar eq 0 then 0 else if Input eq 2 and Avatar eq [x] then [x] else if Input eq 1 then Input eq 1 then (Avatar - 1) else (Avatar + 1);</u>

Sample for Octopus game:

next Avatar == if GameOver or Input eq 0 then Avatar else if Miss eq 1 then 0 else if Input eq 0 then Avatar else if Input eq 1 and Avatar eq 0 then 0 else if Input eq 2 and Avatar eq 6 then 6 else if Input eq 1 then (Avatar - 1) else (Avatar + 1);

Sample for manhole game:

next Avatar == if GameOver or Input eq 0 then Avatar else (Input-1);

3.3. Animation constraint

- Ani[x]Seq[y] variable:
  - if Animation[x] has no trace, Ani[x]Seq[y] should be true when Ani[x] is in step[y]

    Ani[x]Seq[y] == Ani[x] eq [y+1];
  - if Animation[x] has trace, Ani[x]Seq[y] should be true when Ani[x] is in certain range

    Ani[x]Seq[y] == (Ani[x] ge [y+1]) and (Ani[x] le [total_step - y])

- Ani[x] variable:
  - Animations should be only updated when gametick is 0

    if Gametick ne 0 then Ani[x] else ...
  - if miss reset certain animation step, the animation should be reset when miss is triggered; it game is over, animation should be 0

    if ([MissLoc associated with Animation[x]] or Game then 0 else ...
  - if animation is not shown now, it would be shown in the next game tick randomly

    if Ani[x] eq 0 then Random else …
  - if animation is shown now, it should continue to next step

    (Ani[x] + 1) % [Ani[x] step number]

3.4. Miss constraint

- Miss variable: it would be true if either MissLoc[x] is true

  Miss == [or conjunction of MissLoc[x]s variable]

- MissLoc[x] variable: it should be true when the miss condition is trigger ([a],[b],[c] are set in miss configuration)
  - if miss type is “negative”:

    MissLoc[x] == (Ava[a] eq 0) and (Ani[b]Seq[c] eq 1);
  - if miss type is “positive”:

    MissLoc[x] == (Ava[a] eq 1) and (Ani[b]Seq[c] eq 1);

3.5. Score constraint

- if score type is “Avatar Location”, then user score when avatar is in certain location

  <u>Score == (Ava[score location] eq 1) and (Gametick eq 0);</u>

- if score type is “Not Miss”, then user score when animation is in certain step but the corresponding miss condition is not trigger

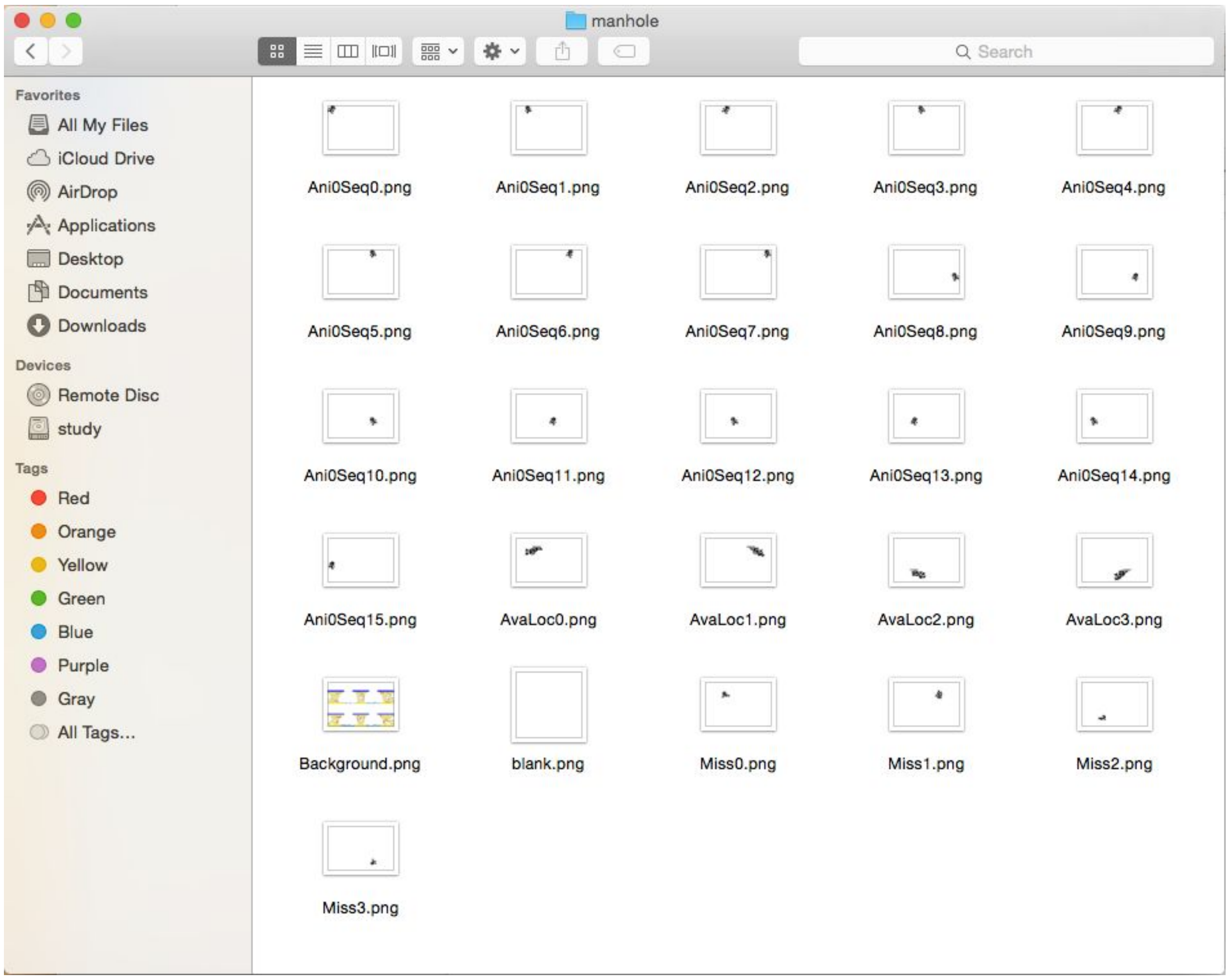
  <u>Score == (Ava[a] eq 0) and (Ani[b]Seq[c] eq 1);</u>

  or

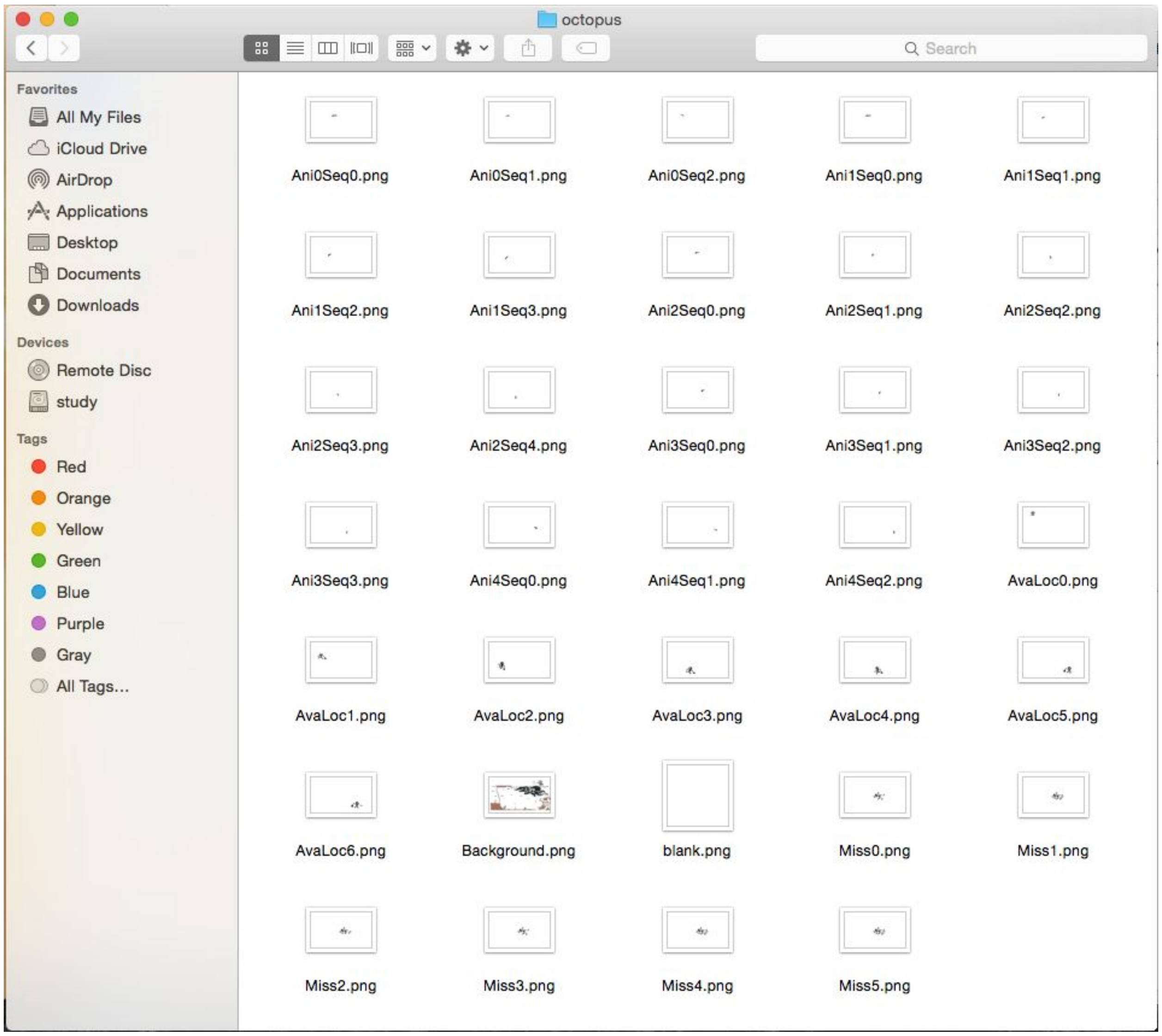
  <u>Score == (Ava[a] eq 1) and (Ani[b]Seq[c] eq 1);</u>

4.Sample Configuration

4.1.Image preparation



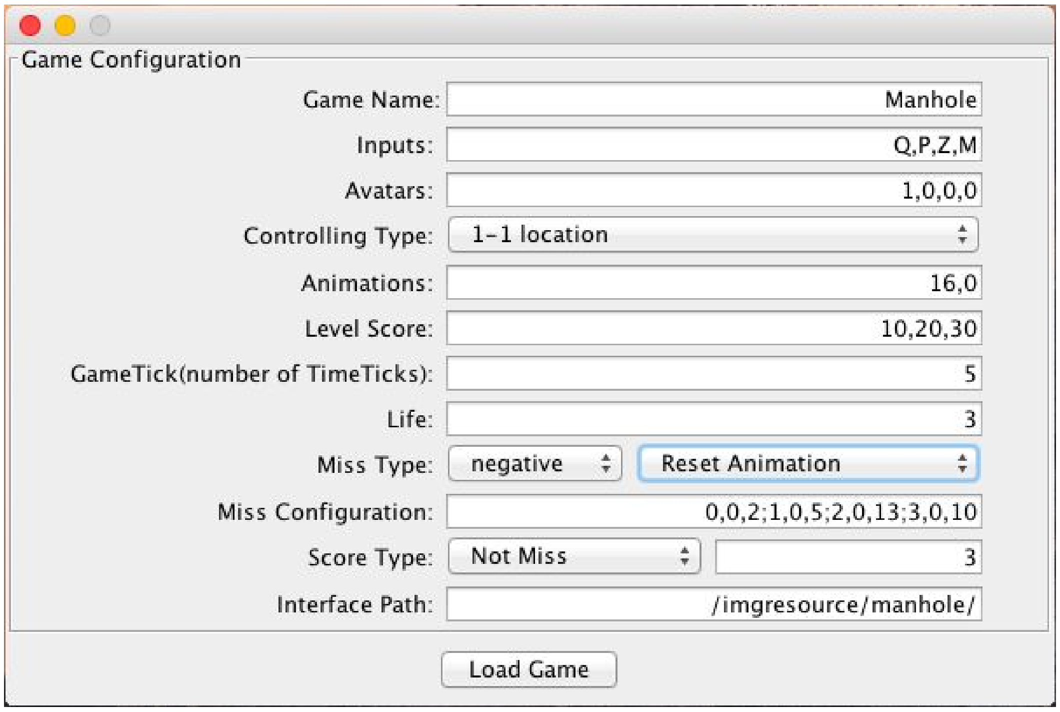
Manhole Images



Ani0Seq0.png Ani0Seq1.png Ani0Seq2.png Ani1Seq0.png Ani1Seq1.png
Ani1Seq2.png Ani1Seq3.png Ani2Seq0.png Ani2Seq1.png Ani2Seq2.png
Ani2Seq3.png Ani2Seq4.png Ani3Seq0.png Ani3Seq1.png Ani3Seq2.png
Ani3Seq3.png Ani4Seq0.png Ani4Seq1.png Ani4Seq2.png AvaLoc0.png
AvaLoc1.png AvaLoc2.png AvaLoc3.png AvaLoc4.png AvaLoc5.png
AvaLoc6.png Background.png blank.png Miss0.png Miss1.png
Miss2.png Miss3.png Miss4.png Miss5.png

Octopus Images

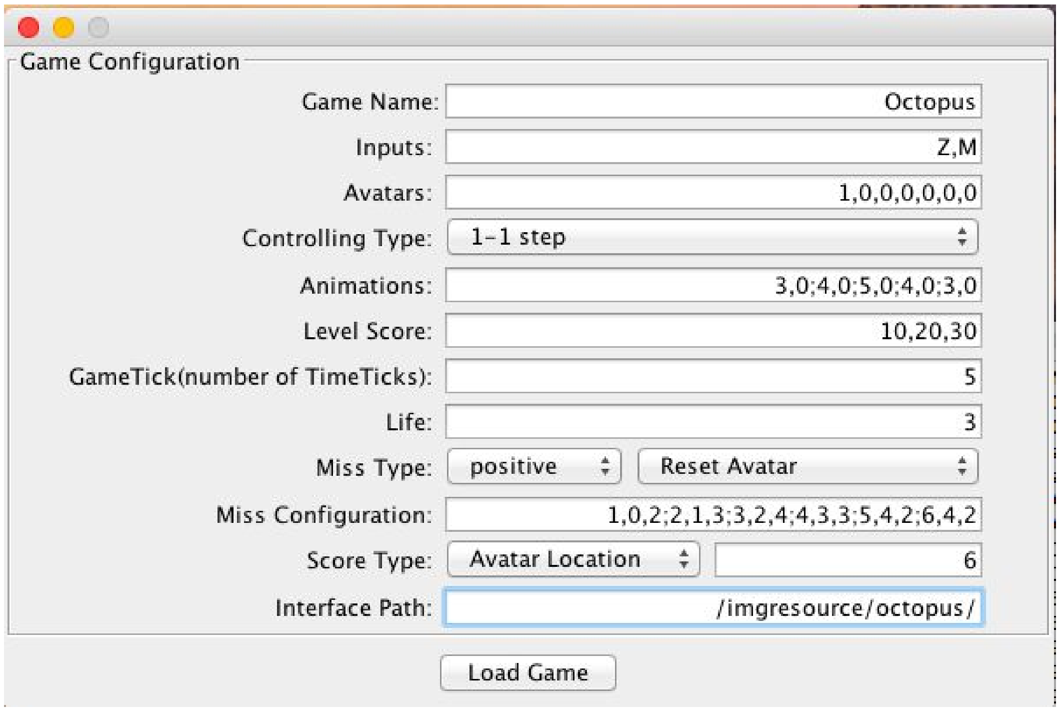
4.2.Game Configuration



Game Configuration

Game Name:	Manhole
Inputs:	Q,P,Z,M
Avatars:	1,0,0,0
Controlling Type:	1-1 location
Animations:	16,0
Level Score:	10,20,30
GameTick(number of TimeTicks):	5
Life:	3
Miss Type:	negative; Reset Animation
Miss Configuration:	0,0,2;1,0,5;2,0,13;3,0,10
Score Type:	Not Miss; 3
Interface Path:	/imgresource/manhole/

Manhole Configuration



Game Configuration

Game Name:	Octopus
Inputs:	Z,M
Avatars:	1,0,0,0,0,0,0
Controlling Type:	1-1 step
Animations:	3,0;4,0;5,0;4,0;3,0
Level Score:	10,20,30
GameTick(number of TimeTicks):	5
Life:	3
Miss Type:	positive; Reset Avatar
Miss Configuration:	1,0,2;2,1,3;3,2,4;4,3,3;5,4,2;6,4,2
Score Type:	Avatar Location; 6
Interface Path:	/imgresource/octopus/

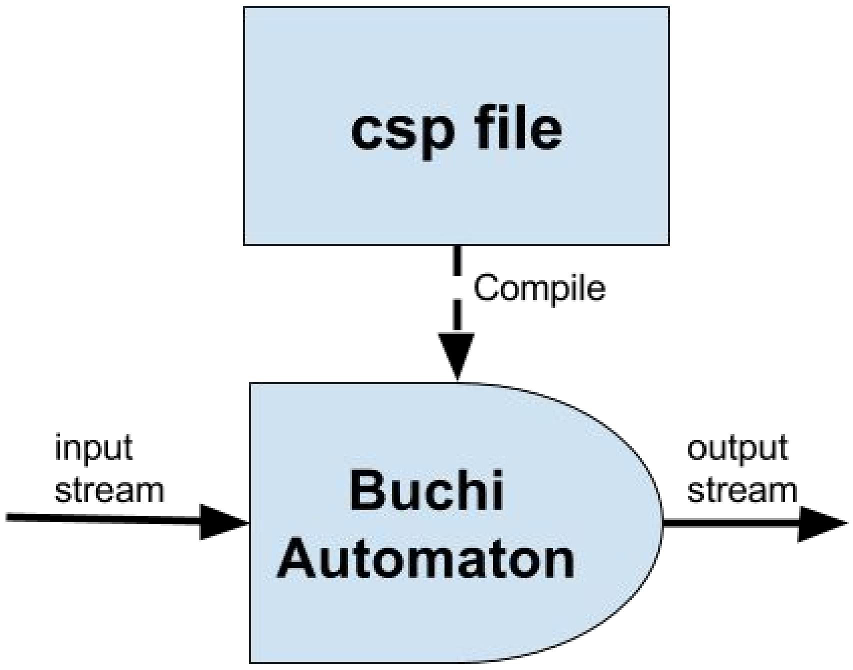
Octopus Configuration

5. Discussion

Here we present some possible ways to reduce the Buchi automaton:

- Free Variable: such as Input and Random
- Periodic Variable: also called self-constraint variable, such as Game tick
- Variable with infinite domain: such as the total score in game and watch
- Duplicated Variable: variables with similar meaning but represent different object, such as Ani[x] variable in “Manhole” game

The execute way of csp code can be:


csp file
Compile
input stream
Buchi Automaton
output stream