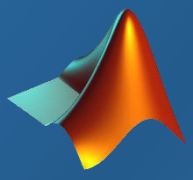
BASICS OF MATHEMATICAL SOFTWARE

MATLAB PROJECT REPORT



Student Name	Luu Tin
Student Number	1700674
Project Name	Snake Game
Date of Submission	December 26th , 2018
Supervisor	Jussi Ojanen

Table of Contents

1. Introduction

1.1. Concept

The “Snake Game” is like anyone knows, a game in which there is a snake trying to eat prey and grow followed by the elevation of level and escalation of corresponding difficulty. Player uses the four navigation arrow buttons to control the moving direction of the snake while using Escape key to quit the game.

1.2. Features

- The game features a 20x20 grids for the snake to move within.
- For each level, the snake will always start with a size of ten grid and a prey is created at the center of the screen.
- Each time the snake gets to eat the prey, a new prey is created randomly in the map and the snake body is lengthen by one grid.
- Every level has requirement needed to level up. Based on which level the player is in, the requirement can vary from simply achieving a certain body length to finishing the level within a limited given amount of time (time-attack feature).
- There is a total of seventy levels. Of which, the first 11 levels (0 -> 10), no time limit is set, but the speed of the snake gradually increases. Then starting from level 11, the player must rush to finish the level before time runs out. Once you finish the last level, you win the game.
- The game is over under either of the two following conditions:
  - The player mistakenly controls the snake to bite itself (including turning around).
  - The player fails to finish the level within the given time limit (for time-attack level).

2. Ideas and Implementation

2.1. Initialization

- The drawing area of MATLAB for this game includes the 20x20 grids for the snake and detail about the situation of the game. The 20x20 grids will be drawn onto a 1000x1000 area, so each grid occupies a 50x50 area. The following code in figure 1 is simply for simulating the implementation of this idea, which is not displayed as one unify block of code in the program. Figure 2 shows the result of this implementation.

```
hold on
plot([0 0 1000 1000 0],[0 1000 1000 0 0], 'k');
for i=50:50:950
    plot([i i],[0 1000],'k--');
    plot([0 1000],[i i],'k--');
end
axis([0 1000 0 1000]);
```

Figure 2. Code for drawing 20x20 grid

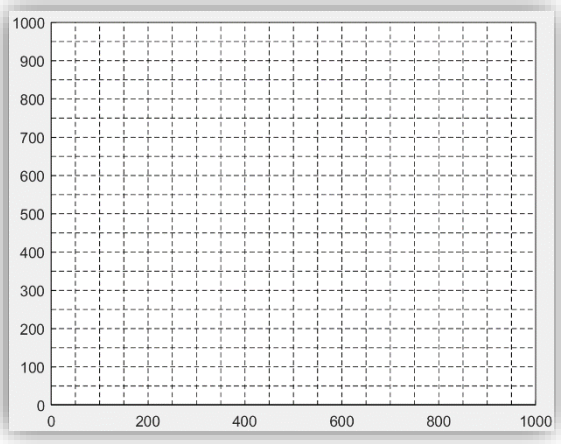


Figure 1. Result for drawing 20x20 grid

- Each grid is determined by its coordinate (1 to 20 for both axes). The one closest to the origin gets a (1,1) coordinate while the furthest gets (20,20).
- Besides, many variables need to be initialized at the first place. Figure 3 shows a list of them.

```
%these two are for keypress action
global key
key = 'rightarrow';
global key_press
key_press=0;
%this check if the game can continue
game_over=0;
%this set the direction for the snake
dir=0;
%this check if the snake has just eaten
eat=0;
%this is to keep track of the score
score=0;
%this is to check if newly-created food is already inside the snake
qualify=0;
%this initializes the position of the prey
food=[10 10];
%the initial size of the snake
size=10;
%this is to check if you have arrived in the passing gate
pass=0;
%this is the limit to pass the level
limit=15;
%this is to change color of the background
col=[1 1 1];
%this is for the time to blink in red and black
status=0;
%this is for the timer to ring
global alarm
alarm=0;
%this is the time limit (only applied pass level 10)
time_limit=35;
%this is the time counter
times=0;
%this indicates which level you are in
level=0; %start with the entry level
%this is to check if you win the game
win=0;
%initialize the position of the snake
x=0:(size-1);
y=zeros([1 size]);
```

Figure 3. Initialization of variables

- According to the listed variables, the initial length of the snake is ten grids. It lies horizontally at the bottom left corner of the drawing area. The prey is located in the center.
- Some of the variables are GLOBAL for the implementation of keypress and timer software interrupt.

2.2. Control and Timing

- As mentioned in the introduction section, the snake is controlled by the four navigation arrow keys, and the Escape key is used to quit the game. So as to achieve this, the keypress software interrupt is employed. The interrupt function informs the program if there is a new key pressed (*key_press* variable) and also returns which key is pressed (*key* variable).

```
function keypress(obj,~)
    %get the key pressed and pass to the main function
    global key
    key=get(obj,'CurrentKey');
    global key_press
    key_press=1;
end
```

Figure 4. Keypress interrupt function

- If an interrupt happens, and the program is notified of the event. Then the program will toggle the *key_press* variable and either properly change the direction of the snake or end the game.

```
if ~win
    if strcmp(key,'rightarrow')
        if dir==1
            title('You bite yourself. Game over.');
            game_over=1;
        else
            dir=0;
        end
    end
    if strcmp(key,'leftarrow')
        if dir==0
            title('You bite yourself. Game over.');
            game_over=1;
        else
            dir=1;
        end
    end
    if strcmp(key,'uparrow')
        if dir==3
            title('You bite yourself. Game over.');
            game_over=1;
        else
            dir=2;
        end
    end
    if strcmp(key,'downarrow')
        if dir==2
            title('You bite yourself. Game over.');
            game_over=1;
        else
            dir=3;
        end
    end
    if strcmp(key,'escape')
        stop(t);
        delete(t);
        return;
    end
end
```

Figure 5. Choices to be made once a key is pressed

- The game features time attack characteristics, so timing is inevitable. This is achieved by using timer software interrupt. The timer is configured to interrupt every one second, and it executes the interrupt function only once. The interrupt function will signal the program of this event (*alarm* variable).

```
t=timer;
t.StartDelay=1;
t.Period=1;
t.TasksToExecute=1;
t.ExecutionMode='fixedRate';
t.TimerFcn=@timerBreak;
start(t);
```

Figure 7. Timer configuration

```
function timerBreak(~,~)
    global alarm
    alarm = 1;
end
```

Figure 6. Timer interrupt function

- As the program receive the *alarm* signal, it increments the *times* variable by one and check if *times* has reached the *time_limit*. Another task is to check the *times* for dimming the screen.

```
if alarm
    alarm=0;
    times=times+1;
    start(t);
    %display(counter);
    if time_limit==times && level>10
        title('\color{white}Out of time. Game over');
        game_over=1;
    end
    if time_limit-times<=30 && level>10
        col=col-[0.025 0.025 0.025];
        fig.Color=col;
        status=1-status;
    end
end
```

Figure 8. Execution for timer timeouts

2.3. Movement and Consumption

- As moving direction has been determined, the next job is to move the snake accordingly. The code in figure 9 is to move the head of the snake forward by one grid given the wall is not hit or go through it to the other side. After that, the remaining part of the snake will advance.

```
if ~pass
    if dir==0
        x(length(x))=x(length(x))+1;
    end
    if x(length(x))>19
        x(length(x))=0;
    end
    if dir==1
        x(length(x))=x(length(x))-1;
    end
    if x(length(x))<0
        x(length(x))=19;
    end
    if dir==2
        y(length(x))=y(length(x))+1;
    end
    if y(length(x))>19
        y(length(x))=0;
    end
    if dir==3
        y(length(x))=y(length(x))-1;
    end
    if y(length(x))<0
        y(length(x))=19;
    end
end
```

Figure 9. Code for moving the snake

- The objective for the player is to control the snake to eat the prey. Thus, as the snake moves, its head will eventually hit the prey sooner or later under the control of the player. Once this happens, the program will execute the code in figure 10. This aims at incrementing the current score by one and creating a new prey at a random position (also check if the new prey happens to be already inside the snake).

```
if (food(1)==x(length(x))) && (food(2)==y(length(x)))
    eat = 1;
    qualify=0;
    while ~qualify
        food = [floor(20*rand(1,1)) floor(20*rand(1,1))];
        qualify=1;
        for i=1:length(x)
            if food(1)==x(i) && food(2)==y(i)
                qualify=0;
                break;
            end
        end
    end
end
```

Figure 10. Code for consuming prey and the creation of new prey

2.4. Score and Growth

As the snake eats more and more, there is one variable tasked of tracking this progress (*score*). Every time, an *eat* happens, the score is increased by one and the snake is expanded by one grid. This is achieved by the code in figure 11.

```
if eat
    score=score+1;
    eat=0;
    x=[x(1) x];
    y=[y(1) y];
    x(2:(length(x)-1))=x(3:length(x));
    y(2:(length(x)-1))=y(3:length(x));
else
    x(1:(length(x)-1))=x(2:length(x));
    y(1:(length(x)-1))=y(2:length(x));
end
```

Figure 11. Code for tracking progress and expansion of the snake

2.5. Leveling-up

- Once the snake eats enough and is ready to advance to a new level, a gate is opened. This is achieved in *Graphics* section. As the snake reaches the gate, its body will gradually disappear. Code in figure 12 materializes this idea.

```
if length(x)>=limit && x(length(x))==0 && ...
        y(length(x))>=8 && y(length(x))<=11
    x(length(x))=-1;
    y(length(x))=-1;
    pass=1;
end
```

Figure 12. Vaporizing the snake into thin air while advancing level

- Once a new level is reached, the program will update details for the new level. This includes but not limited to resetting length, location of snake and prey, updating time limit and requirement for leveling up. This is done by the code in figure 13.

```
if x(1)==-1
    pass=0;
    level=level+1;
    x=0:(size-1);
    y=zeros([1 size]);
    dir=0;
    food=[10 10];
    times=0;
    col=[1 1 1];
    fig.Color=col;
    status=0;
    if level==71
        win=1;
    end
    if level>60
        limit=80;
        time_limit=800-35*(level-60);
    end
    if level<=60 && level>50
        limit=70;
        time_limit=700-30*(level-50);
    end
    if level<=50 && level>40
        limit=60;
        time_limit=600-25*(level-40);
    end
    if level<=40 && level>30
        limit=50;
        time_limit=500-20*(level-30);
    end
    if level<=30 && level>20
        limit=40;
        time_limit=400-15*(level-20);
    end
    if level<=20 && level>10
        limit=30;
        time_limit=300-10*(level-10);
    end
    if level<=10 && level>5
        limit=30;
    end
    if level<=5
        limit=20;
    end
end
```

Figure 13. Updating details for new level

- One last thing is about the speed of the snake, figure 14 shows the code that uses pause to implement the delay for each step.

```
if level<10
    pause(0.03-level*0.003); %pause for a while
else
    pause(0.002);
end
%check if game is over
if game_over
    stop(t);
    delete(t);
    break;
end
```

Figure 14. Code for speed of the snake

2.6. Graphics

- First, code in figure 15 draws the border of the drawing area and the gate reaching new level.

```
hold on;
%draw the border of the drawing area
plot([0 0 1000 1000 0],[0 1000 1000 0 0], 'k');
%if pass the level
if pass
    plot([0 0 50 50 0],[400 600 600 400 400], 'k');
end
```

Figure 15. Code for drawing border and gate

- Every time a prey is consumed, a new one gets created, the code in figure 16 does this job.

```
%draw prey for the snake
fill([food(1)*50+26 food(1)*50+51 food(1)*50+26 food(1)*50+1]...
    ,[food(2)*50+1 food(2)*50+26 food(2)*50+51 food(2)*50+26],'r');
```

Figure 16. Code for drawing the prey

- Then figure 17 depicts the code for drawing the head and the body of the snake.

```
%draw the snake except its head
for i=1:(length(x)-1)
    fill([x(i)*50+1 x(i)*50+51 x(i)*50+1 x(i)*50+51]...
        ,[y(i)*50+1 y(i)*50+1 y(i)*50+51 y(i)*50+51],'b');
end
%draw the snake's head
i=0:(pi/500):(2*pi);
X=x(length(x))*50+25+25*cos(i);
Y=y(length(x))*50+25+25*sin(i);
fill(X,Y,'b');
```

Figure 17. Code for drawing the snake

- Next, displaying the score, requirement for level up, time elapsed and also changing the time display color is achieved thanks to the code in figure 18.

```
text(0, 1025,sprintf('Score %d/%d', length,limit));
if level<10
    text(900, 1025,sprintf('Level %d', level));
else
    text(880, 1025,sprintf('Level %d', level));
end
if times/60>=10 && mod(times,60)>=10
    h=text(0,-25,sprintf('Times elapsed %d:%d', floor(times/60),mod(times,60)));
end
if times/60<10 && mod(times,60)>=10
    h=text(0,-25,sprintf('Times elapsed 0%d:%d', floor(times/60),mod(times,60)));
end
if times/60>=10 && mod(times,60)<10
    h=text(0,-25,sprintf('Times elapsed %d:0%d', floor(times/60),mod(times,60)));
end
if times/60<10 && mod(times,60)<10
    h=text(0,-25,sprintf('Times elapsed 0%d:0%d', floor(times/60),mod(times,60)));
end
if status
    h.Color = [1 0 0];
else
    h.Color = [0 0 0];
end
```

Figure 18. Code for displaying some simple details and dimming the screen

- And then when it comes to display time limit for the level this task is performed by the code in figure 19.

```
if level<=10
    text(780,-25,'Time limit 00:00');
else
    if time_limit/60>=10 && mod(time_limit,60)>=10
        text(780,-25,sprintf('Time limit %d:%d', floor(time_limit/60),mod(time_limit,60)));
    end
    if time_limit/60>=10 && mod(time_limit,60)<10
        text(780,-25,sprintf('Time limit %d:0%d', floor(time_limit/60),mod(time_limit,60)));
    end
    if time_limit/60<10 && mod(time_limit,60)>=10
        text(780,-25,sprintf('Time limit 0%d:%d', floor(time_limit/60),mod(time_limit,60)));
    end
    if time_limit/6<10 && mod(time_limit,60)<10
        text(780,-25,sprintf('Time limit 0%d:0%d', floor(time_limit/60),mod(time_limit,60)));
    end
end
```

Figure 19. Code for displaying time limit for the current level

- Last but not least, in order to dimming the background, the code in figure 20 is used.

```
if time_limit-times<=30 && level>10
    col=col-[0.025 0.025 0.025];
    fig.Color=col;
    status=1-status;
end
```

Figure 20. Code for dimming the background

3. Result

- Figure 21 depicts the initialization of the game.

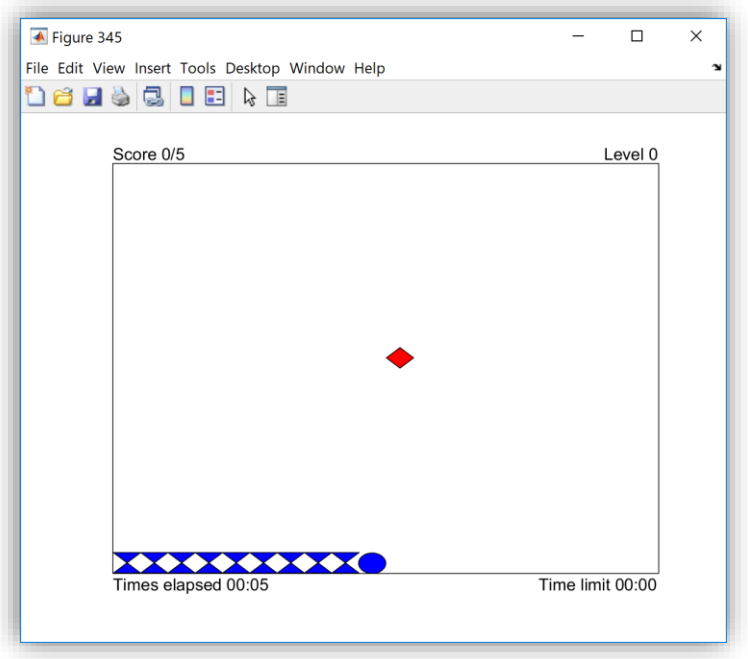


Figure 21. Initialization of the game

- Figure 22 shows the snake is grown by four grids and trying to eat the next prey.

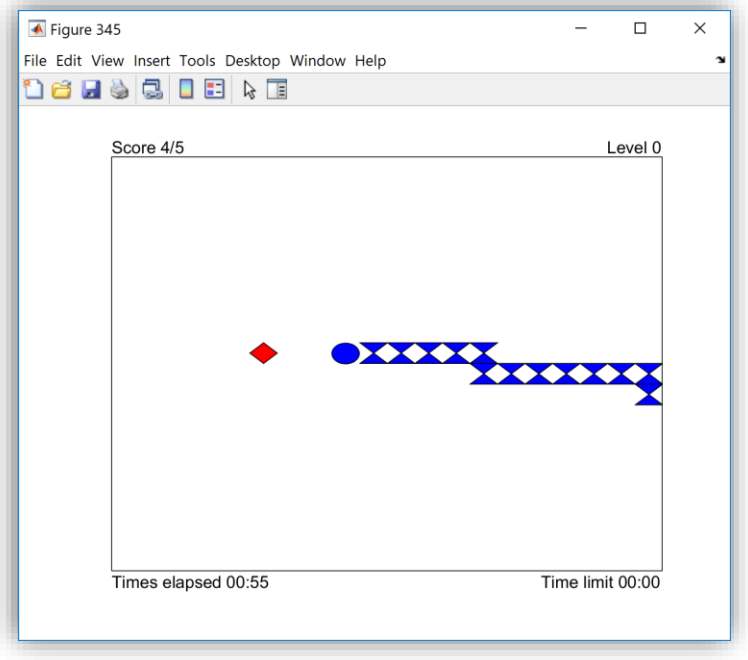


Figure 22. The growth of the snake and its approach toward the prey

- Figure 23 demonstrates the level up circumstance.

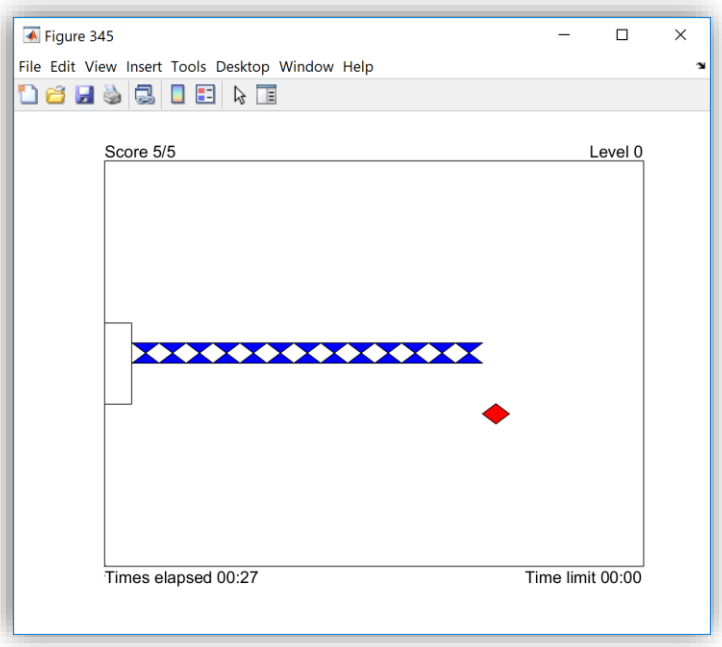


Figure 23. The emergence of level-up gate and the vanishing of the snake

- Figure 24 features the snake biting itself and ends the game.

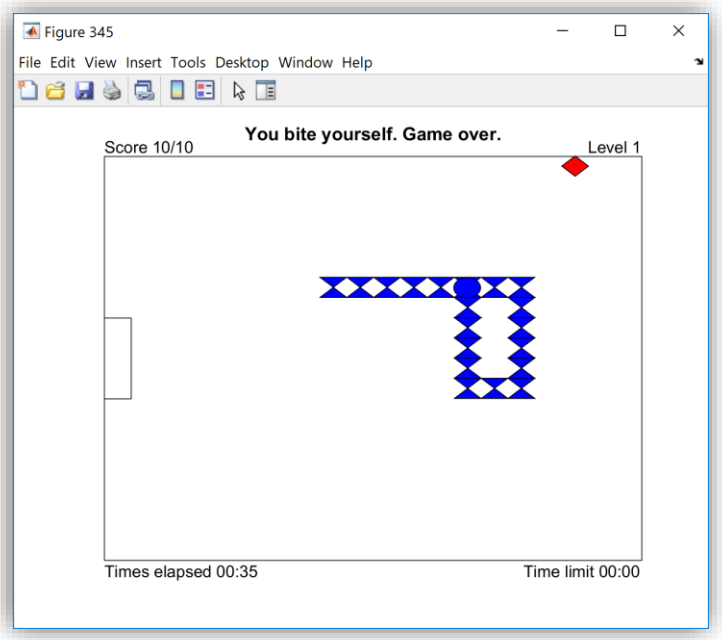


Figure 24.Game is over

- Other features are better experienced by real-time playing because static photos cannot fully demonstrate them.

4. Conclusion

The Snake Game in general achieves what a generic game of this type needs to be; however, due to various factor, the performance of the game is still debatable. Besides, there are still room for development of graphical properties at the expense of further performance loss. Moreover, there are pending features that may be added in the future. All in all, this game is no perfect version for a snake game as flaws remain everywhere, but at the very least, it has achieved the basics of what it needs to.