Hong Kong Diploma of Secondary Education Examination

School Based Assessment

Information and Communication Technology

Option D: Software Development

Topic: Puzzle & mini games

School: Cheung Sha Wan Catholic Secondary School

Name: Tang Hadrian Wai To

Class: 6B

Class number: 30

**Chapter 1: Preliminary Investigation and System Analysis**

**1.1: Background**

Nowadays, in this technologically-driven world, children are exposed to various kinds of entertainment such as video games and movies online. However, as the Internet is a free place where anyone can share anything, including some content deemed immoral or obscene by the general public, such as violent games and pornographic films, these children are in need of some games healthier to their minds.

In line with the worldwide trend of protecting children’s use of computers, such as the $170M fine over children’s privacy for YouTube by the U.S. Federal Trade Commission, I have developed a program that provides safer entertainment for children by providing them fun-to-play puzzle games.

**1.2: Aim**

To provide 3 different fun-to-play games that can automatically save progress, in order to train children’s logical and critical thinking alongside providing entertainment so that children can be willing to continue playing. The key aim is being fun to play, which is most suitable for children.

**1.3: Target Users**

This solution is aimed at lonely and bored people who wish for exciting games with moderate difficulty. Being easy to open and operate, this solution will be a good choice for them.

Children are another target. Being in need of exciting games that are non-violent, this can provide relief from homework and tests for them.

**1.4: Objectives**

1. Include 3 different games
2. Include clear and concise instructions for each game
3. Include user login
4. Include user registration
5. Include automatic progress saving
6. Include beautiful visuals to attract children’s attention
7. Include elements of fun

**Chapter 2: System Design**

**2.1: The User Interface**

This program runs on the Command Line Interface (CLI) instead of the Graphical User Interface (GUI), as it uses less system resources, as the program is text-based and does not require complex graphical processing, and is more cross-platform, as all computer operating systems have one. By detecting key presses instead of commands for user input and providing clear instructions on screen, the disadvantage of CLI where the user has to remember commands to input can be avoided. Moreover, using CLI, the overall aesthetics can be made more consistent as all text use the same font and the players can feel the beauty of consistency. The player will be more engaged in playing these games, thus increasing emotional responses and increasing the effectiveness of such system.

**2.2: Modularization**

In this program, there are 5 “rooms” where the controlled character can move through; of which 3 are games to play. Each room is a module where the player can navigate through different situations and obstacles, which increase the excitement and fun of the games. When the player successfully completes a room, the progress is automatically saved so that the player does not have to go through past progress again.

**2.3: Diagrams**

The following outline the design of the system, shown based on the functions and objectives stated in Chapter 1.

The following will be shown:

1. The Level 0 data flow of the system

2. The Level 1 data flow of the system

3. The Level 2 data flow of the system

4. The Structure Chart of the system

5. The System Flowchart of whole system

6. The System Flowchart of 3 Games

**2.4: The Level 0 data flow of the system**

The system consists of the puzzle game program and the player.

Player

Game input

Game state

Puzzle Game Program

Save user information

Loaded user information

D1 User information

**2.5: The Level 1 data flow of the system**

A more detailed flow can be found from this level 1 data flow diagram.

2.0

Registration Process

6.0

Connect Four Process

4.0

Number Guess Process

3.0

Title Screen Process

1.0

Login Process

D1 User information

ID & Password

Player

ID & Password Login information Registration Confirmation

Player

User IDs

New registration account

New progress

7.0

Winning Screen Process

New progress

New progress

New progress

New progress

5.0

Bulls and Cows Process

Avatar  
control Avatar control Game

Player

Avatar control and piece state

Avatar Game and number input placement  
movement state

Player

Player

Game  
 state

The details of sub-programs in the level 1 data flow diagram of the system are as follows.

|  |  |
| --- | --- |
| Sub-program | Description |
| 1.0  Login Process | This process allows user to input his/her ID and Password into the system. Then, the process will retrieve users' information from D1 data so as to do verification. |
| 2.0  Registration Process | This process allows user to input his/her ID and Password for registration. The process will then send new user's information to D1 data and send registration results to user. |
| 3.0  Title Screen Process | This process requires user to input his/her control of the avatar in-game. The process will then return the game state to the user. When this part is completed, the updated progress of the game will be saved to D1 data. |
| 4.0  Number Guess Process | This process requires user to input his/her guesses of the generated random number in-game. The process will then show the game state including whether the guess is smaller/greater than the actual number in game to user. If the guess is correct, then the gate blocking the way will open. When this part is completed, the updated progress of the game will be saved to D1 data. |
| 5.0  Bulls and Cows Process | This process requires user to input his/her guess of the code in the game. The process will then return the tips and the guessing results in-game to user. When this part is completed, the updated progress of the game will be saved to D1 data. |
| 6.0  Connect Four Process | This process requires user to input his/her choice of piece placement in the game. The process will then decide the opponent input and game state in-game to user. When this part is completed, the updated progress of the game will be saved to D1 data. |
| 7.0  Winning Screen Process | This process acts as the “prize” for the player and displays fireworks of different colors to congratulate the player. The player can choose to return to Title Screen Process and start again, where the updated progress of the game will be saved to D1 data. |

The details of the data file in the level 1 data flow diagram of the system are as follows.

|  |  |
| --- | --- |
| Database File | Description |
| D1  Data | This database file stores all the users' information, which contains users' ID, password and progress. |

**2.6 The level 2 data flow diagram of different processes**

A more detailed flow can be found from the following level 2 data flow diagrams.

Process 1.0:

1.1

Input

ID & Password

D1 Data

Player

ID & Password User Information

Verification Result

1.2

Verification

Process 2.0:

Player

2.3

Registration

2.2

Validation

2.1

Input ID

ID

ID

ID of all players

D1 Data

Password Validation flag

Registration result Player’s account

Process 3.0:

Move command

3.2

Move avatar

3.3

Update screen

3.1

Interpreting user input

Player

User input

New location

New progress

New screen

D1 Data

Process 4.0:

4.1

Interpreting user input

User input

D1 Data

Move command Add/Remove digit Submit command

New progress command

4.4

Compare input number and actual number

4.3

Update inputted number

4.2

Move avatar

Player

New screen Comparison result

New location Updated number

4.5

Remove obstacle

Updated game state

4.6

Update screen

Process 5.0:

5.1

Interpreting user input

User input

D1 Data

Move command Add/Remove digit Submit command

New progress command

5.4

Compare individual digits and their locations between input and actual

5.3

Update inputted number

5.2

Move avatar

Player

New screen Difference

New location Updated number

5.5

Remove obstacle

Updated game state

5.6

Update screen

Process 6.0:

6.1

Interpreting user input

User input

D1 Data

Move command Location to place a piece

New progress

6.5

Checking whether a win occurred

6.3

Player piece placement

6.2

Move avatar

Player

Win flag

New screen Location to place a piece Winning Player

New location

6.4

CPU piece placement

Location of pieces

Updated game state

6.6

Update screen

Process 7.0:

7.1

Interpreting user input

User input

7.4

Firework progression

D1 Data

Move command

New progress

Firework location

7.5

Firework removal

7.2

Move avatar

Player

7.3

Spawn firework

New screen

New location Firework Removed

New firework location firework

Random available Next  
 location sprite

7.6

Update screen

**2.7 The structure chart of the system**

A detailed structure of the system is shown in this chart.

Puzzle game system

User registration

Reading and writing to data file

Login system

IDs and passwords IDs and  
 passwords

User information

Authenticated flag

User login

Main game

Winning Screen

Connect Four

Bulls and Cows

Number Guessing

Title screen

Key press

Update game state

New game state

**2.8 The system flowchart of the whole system**

The flow of the whole system will be shown in the chart.

Start

Key press

Update game state

Data file

Save

Updated game state

Game

Data file

Check existence

Verification

ID & Password

ID & Password

Registration or Login

Registration ` Login

**2.9 The system flowchart of Number Guess**

This is the detailed system flowchart of the Number Guess game.

Next game

Number is too large

Data file

Save progress

Control avatar to move to the next room

Remove barrier

Number is correct

Number is too small

Compare input and actual number

Game

Enter guess

Input < Actual Input = Actual Input > Actual

**2.10 The system flowchart of Bulls and Cows**

This is the detailed system flowchart of the Bulls and Cows game.

Enter guess

Game

Check guess against answer

Trials and AnswerTrials and Answer Trials and Answer Trials and AnswerTrials and Answer Trials and AnswerTrials

Guess is incorrect Guess is correct

Display tips for this guess

Number is correct

Remove barrier

Control avatar to move to the next room

Save progress

Data file

Next game

**2.11 The system flowchart of Connect Four**

This is the detailed system flowchart of the Connect Four game.

Game

Enter piece placement position

Check horizontals

Exit and re-enter the room

CPU wins No one wins Player wins

Check verticals

CPU wins No one wins Player wins

CPU places piece at this position

CPU places piece at random position

You win!

Remove barrier

Control avatar to move to the next room

Save progress

Next game

You lose!

Check horizontals

Check verticals

Check diagonals

Simulate all 7 possible positions

Check horizontals

Check verticals

Check diagonals

You lose!

Check diagonals

CPU wins Player wins

No one wins

CPU/Player wins No one wins

CPU/Player wins No one wins

Data file

CPU/Player wins No one wins

CPU wins Player wins

No one wins Player wins

CPU wins

CPU wins No one wins Player wins

No one wins

**Chapter 3: System Implementation**

**3.1: Summary**

In this chapter, the implementation of different aspects of the system will be shown, including:

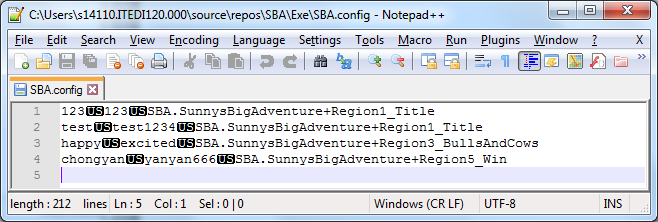
1. Database Implementation

2. User Interface Implementation

3. Process Implementation

**3.2: Database implementation**

The system consists of one data file only, named “SBA.config” with a structure as follows:



Each line is a record, where individual fields are separated by the Unit Separator (U+001F). The user’s ID is stored in the first field, the password is stored in the second field, and the progress is serialized to the third field.

The reason the Unit Separator is used instead of

1. Line breaks: The file can be viewed by an appropriate file viewer more easily like Notepad++, where each line break defines a record.
2. Commas or tabs: Users can input these characters easily and may corrupt the file.

Whenever the file is read, the framework-provided function System.IO.File.ReadAllLines is used to read the file to a string array, where each line break is interpreted as a record separator.

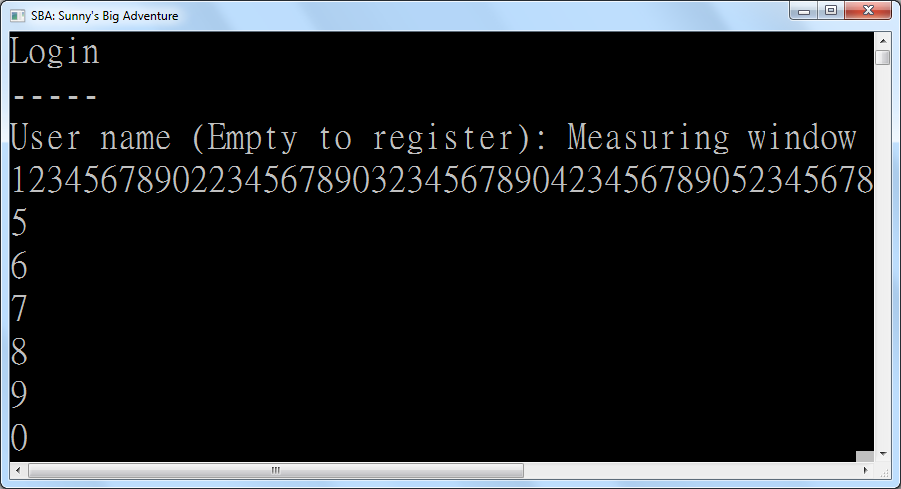
When the array is searched, each string will be split by the Unit Separator and the relevant field will be extracted and used.

If a field needs to be updated, the updated field will be combined with other unchanged fields through the Unit Separator and replace the old record in the array. The array will be saved back to the file through the framework-provided function System.IO.File.WriteAllLines.

The array will never be saved in the program memory to ensure that there is only one source of truth: the file, to avoid bugs arising from inconsistencies between the file and the program memory. Other than that, this approach can also ensure a more pleasant debugging experience where changes in the data file will be reflected in the program immediately, without having to restart the program. Moreover, program crashes will have a minimal chance of causing data loss due to not saving user data in time.

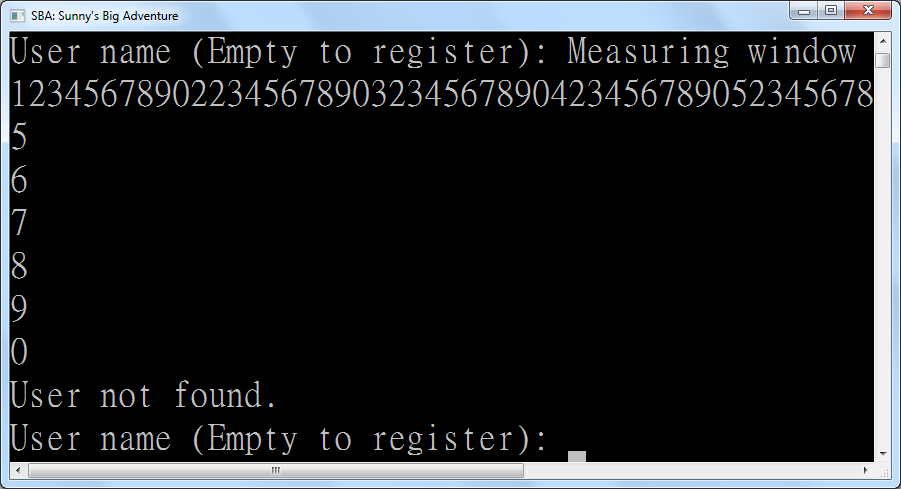
**3.3: User Interface implementation**

The program’s user interface consists of one single console window with a width of 48 blocks and a height of 10 blocks. It has a title of “SBA: Sunny’s Big Adventure” to indicate clearly that this program is a game.

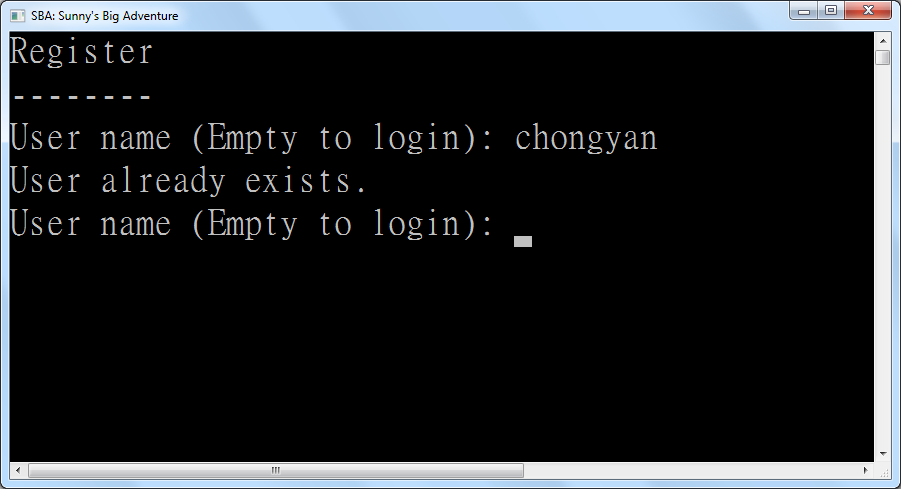


The console window is used to give instructions and guidelines to its users. Users can simply follow the instructions and guidelines given by the system and input movement keys or input keys in order to enjoy the game.

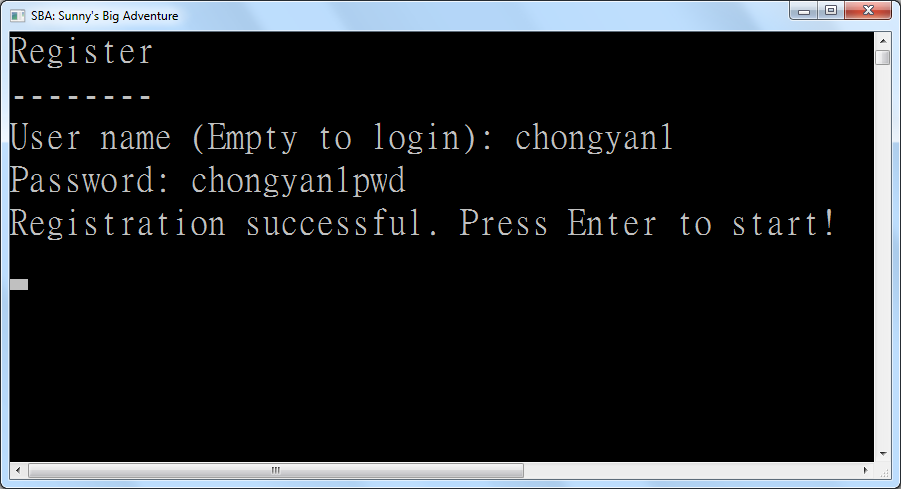
If errors related to invalid input occur, namely “User not found” and “Incorrect password” for login, as well as “User already exists” for registration, that error will be displayed to the user for inputting the data again. This will continue forever until the user finally provides valid data.



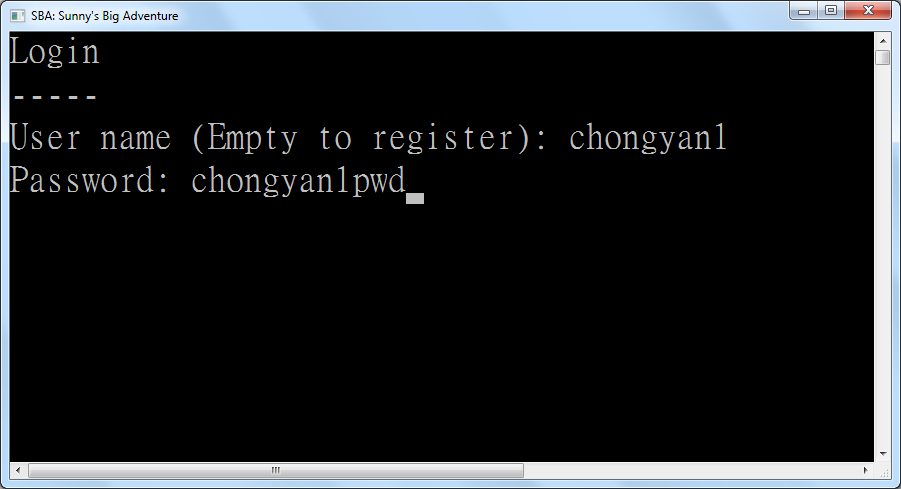
To eliminate the use of user-unfriendly “y” or “n” inputs, empty input is treated as the command for switching between login and registration screens. This way, less time is spent on inputting decisions that the user rarely inputs, like whether to register a new account, as each person usually only ever registers one account only. This way, user satisfaction is enhanced by making the usual path easy to take.



*The error for “User already exists”*



*Successful registration*



*Successful login*

**3.4: Data structure implementation**

Implementation of the processes can be described through the explanation of every structure and class. The function and characteristics of every structure and class will be mentioned one by one in this section.

Structure Delta(Of T)

Sub New(unchanged As T)

Changed = False

OldValue = unchanged

NewValue = unchanged

End Sub

Sub New(oldValue As T, newValue As T)

Changed = True

Me.OldValue = oldValue

Me.NewValue = newValue

End Sub

Public ReadOnly Property Changed As Boolean

Public ReadOnly Property OldValue As T

Public ReadOnly Property NewValue As T

End Structure

The Delta structure represents a change in what to show on screen. It contains fields for the old value and the new value of any data that are scheduled to be updated. For example, it is used for updating sprites and position of each entity.

Structure Point

Implements IComparable(Of Point)

Public Sub New(left As Integer, top As Integer)

Me.Left = left

Me.Top = top

End Sub

Public ReadOnly Property Left As Integer

Public ReadOnly Property Top As Integer

Public Overrides Function ToString() As String

Return $"({Left}, {Top})"

End Function

Public Function CompareTo(other As Point) As Integer Implements IComparable(Of Point).CompareTo

If Left < other.Left Then Return -1

If Left > other.Left Then Return 1

If Top < other.Top Then Return -1

If Top > other.Top Then Return 1

Return 0

End Function

Public Shared Operator =(p1 As Point, p2 As Point) As Boolean

Return p1.Left = p2.Left AndAlso p1.Top = p2.Top

End Operator

Public Shared Operator <>(p1 As Point, p2 As Point) As Boolean

Return p1.Left <> p2.Left OrElse p1.Top <> p2.Top

End Operator

End Structure

This structure represents a point on screen. The coordinate system used starts from the top left and the x-coordinate increments towards to the right while the y-coordinate increments downwards. This is used for entity positioning, for example, rectangular boxes and the player on screen.

Structure Rectangle

Public Sub New(left As Integer, top As Integer, width As Integer, height As Integer)

Me.New(New Point(left, top), width, height)

End Sub

Public Sub New(topLeft As Point, bottomRight As Point)

Me.New(topLeft, bottomRight.Left - topLeft.Left + 1, bottomRight.Top - topLeft.Top + 1)

End Sub

Public Sub New(topLeft As Point, width As Integer, height As Integer)

Me.TopLeft = topLeft

Me.Width = width

Me.Height = height

End Sub

Public ReadOnly Property TopLeft As Point

Public ReadOnly Property Width As Integer

Public ReadOnly Property Height As Integer

Public ReadOnly Property Left As Integer

Get

Return TopLeft.Left

End Get

End Property

Public ReadOnly Property Right As Integer

Get

Return TopLeft.Left + Width - 1

End Get

End Property

Public ReadOnly Property Top As Integer

Get

Return TopLeft.Top

End Get

End Property

Public ReadOnly Property Bottom As Integer

Get

Return TopLeft.Top + Height - 1

End Get

End Property

Function PreciseCollidesWith(other As Rectangle) As Boolean

Return Left <= other.Right AndAlso other.Left <= Right AndAlso Top <= other.Bottom AndAlso other.Top <= Bottom

End Function

Function SafeCollidesWith(other As Rectangle) As Boolean

Return Left - 1 <= other.Right AndAlso other.Left - 1 <= Right AndAlso Top <= other.Bottom AndAlso other.Top <= Bottom

End Function

Public Overrides Function ToString() As String

Return $"({Left}, {Top}) to ({Right}, {Bottom})"

End Function

End Structure

The Rectangle structure represents a rectangle on screen, containing the top-left point, width and height of the rectangle. It also has object methods to help determine whether two rectangles collide. The precise version is used for determining the usable area of trigger zones (areas where key presses can initiate actions) whereas the safe version is used for entity collision to avoid overdrawing due to player movement on screen.

Structure Sprite

Public Sub New(display As Char, Optional color As ConsoleColor = ConsoleColor.White) ' Consoles don't support surrogate pairs

Me.Display = display

Me.Color = color

End Sub

Public ReadOnly Property Display As Char

Public ReadOnly Property Color As ConsoleColor

End Structure

The Sprite structure is a colored character. This represents what a single-character entity can be displayed as. This structure can facilitate maintainability as all sprites are defined in one place.

**3.5: Implementation of helper functions and properties**

Property CursorPosition As Point

Get

Return New Point(CursorLeft, CursorTop)

End Get

Set(value As Point)

SetCursorPosition(value.Left, value.Top)

End Set

End Property

The CursorPosition property is a convenient way to set the current cursor position in the console. With the Base Class Library, CursorLeft (the x-coordinate) and CursorTop (the y-coordinate) need to be set explicitly each time. By setting them both in a single helper property, the chance of bugs due to forgetting to set one component of the coordinate is reduced.

Sub IfHasValue(Of T As Structure)(nullable As T?, f As Action(Of T))

If nullable.HasValue Then f(nullable.GetValueOrDefault())

End Sub

Function IfHasValue(Of T As Structure, TReturn As Structure)(nullable As T?, f As Func(Of T, TReturn)) As TReturn?

Return If(nullable.HasValue, f(nullable.GetValueOrDefault()), New TReturn?())

End Function

Function IfHasValue(Of T As Structure, TReturn As Structure)(nullable As T?, f As Func(Of T, TReturn?)) As TReturn?

Return If(nullable.HasValue, f(nullable.GetValueOrDefault()), New TReturn?())

End Function

Function IfHasValue(Of T As Structure, TReturn)(nullable As T?, f As Func(Of T, TReturn), defaultValue As TReturn) As TReturn

Return If(nullable.HasValue, f(nullable.GetValueOrDefault()), defaultValue)

End Function

The IfHasValue functions are convenient ways to return a value based the non-null value of nullable structures if available. If not, a default value can be supplied. This can reduce lines of code acting on, for example, nullable positions of entities. If the position of an entity is null (not displayed on screen), the code need not draw it again, improving the efficiency of the program.

Function ReadKey(timeout As TimeSpan) As ConsoleKey?

If KeyAvailable Then Return Console.ReadKey(True).Key

Dim beginWait = Date.Now

While Not KeyAvailable And Date.Now.Subtract(beginWait) < timeout

Threading.Thread.Sleep(100)

If KeyAvailable Then Return Console.ReadKey(True).Key

End While

Return Nothing

End Function

The ReadKey function is a convenient way to read a key from the user with a timeout. This is used in the main program where the game updates the position of entities based on whether the entities are mid-air if the player does not provide a keyboard input to move the avatar.

Sub WriteAt(position As Point?, sprite As Sprite)

If position.HasValue Then

CursorPosition = position.GetValueOrDefault()

ForegroundColor = sprite.Color

Write(sprite.Display)

ResetColor()

End If

End Sub

The WriteAt subroutine is a convenient way to write a sprite on screen, fully respecting the character and the color of the sprite. This prevents bugs, for example, from forgetting to account for the color of the sprite, and increases maintainability by achieving code reuse. The reason this subroutine is not a method in the Sprite structure is that this function modifies the console window, thus is not a suitable candidate for bundling with the structure, which should be immutable, i.e. only store data and any methods should have the same outcome when called.

ReadOnly Random As New Random()

<Runtime.CompilerServices.Extension>

Function RandomItem(Of T)(list As ICollection(Of T)) As T

Return list.ElementAt(Random.Next(list.Count))

End Function

The RandomItem function picks a random item from a collection of items. This is used in the Connect Four AI when it randomly chooses a choice from filtered choices.

Const FileName = "SBA.config"

Const FieldSeparator = ChrW(&H1F) ' U+001F Unit Separator

Dim UserName As String

These are universal fields and constants throughout the game. The constants can achieve code reuse as well as ease of updating the save configuration. The UserName field being at the global level instead of only in the main program is due to it being used for looking up the corresponding record to update in the save file whenever the user makes progress.

Sub SaveRegion(region As Region)

Dim config = IO.File.ReadAllLines(FileName)

For i = 0 To config.Length - 1

Dim parts = config(i).Split(FieldSeparator)

If parts(0) = UserName Then \_

config(i) = String.Join(FieldSeparator, parts(0), parts(1), region.GetType().FullName)

Next

IO.File.WriteAllLines(FileName, config)

End Sub

The SaveRegion subroutine is used for storing the updated progress whenever the user enters a new room. This also contains the lookup code for finding the corresponding record to update based on the current User Name.

**3.6: Implementation of entity classes**

The following is the list of all possible entities in the game. An entity is a logical object that can interact with the player. Apart from the login and registration screens, all other screens in this game are made up of entities.

**3.6.1: Base class - Entity**

MustInherit Class Entity

Implements IDisposable

Sub New(entities As ICollection(Of Entity))

entities.Add(Me)

End Sub

This is the entity constructor. A collection is provided in order to add itself into the collection for use in determining entity collision.

Protected Overridable Function ForbidEntry(other As Entity, otherBounds As Rectangle) As Boolean

Return TypeOf other IsNot TriggerZone AndAlso

Bounds IsNot Nothing AndAlso

Bounds.GetValueOrDefault().SafeCollidesWith(otherBounds)

End Function

The ForbidEntry function determines whether other entities can be moved into this entity. It is overridable because instances of the trigger zone class, an entity subclass, are not shown as physical objects and other entities can move freely inside it. Thus, they will override this function to allow other entities to move inside them.

Protected Function CanMoveTo(value As Rectangle?) As Boolean

If value IsNot Nothing AndAlso CurrentRegion IsNot Nothing Then

Dim rect = value.GetValueOrDefault()

If rect.Left < 0 OrElse rect.Right >= WindowWidth OrElse rect.Top < 0 OrElse rect.Bottom >= WindowHeight Then Return False

Dim newPosition = Position

For Each entity In CurrentRegion.Entities

If Me IsNot entity AndAlso entity.ForbidEntry(Me, rect) Then Return False

If newPosition <> Position Then Return False ' Position was set in ForbidEntry, already moved elsewhere

Next

End If

Return True

End Function

The CanMoveTo function determines whether this entity can move to a new position by looping through entities in the current region. If the entity will move out of bounds, then the entity cannot move towards that position. If an entity in the current region forbids this entity from moving into it, then this entity will fail to move to the new position. Also, as other entities can move the current entity’s position in ForbidEntry, this function will need to check if this entity’s position was modified to not move it back to its original position.

Protected MustOverride Sub RedrawAt(bounds As Delta(Of Rectangle?))

The RedrawAt subroutine is marked as MustOverride as each entity has its own appearance and cannot be factored into common code. As a result, the subclasses will have their own implementation of this subroutine.

Dim \_bounds As Rectangle?

Protected Property Bounds As Rectangle?

Get

Return \_bounds

End Get

Set(value As Rectangle?)

If Not CanMoveTo(value) Then Return

RedrawAt(New Delta(Of Rectangle?)(\_bounds, value))

\_bounds = value

End Set

End Property

The Bounds subroutine is marked as MustOverride as each entity has its own appearance and cannot be factored into common code. As a result, the subclasses will have their own implementation of this subroutine.

Public Property Position As Point?

Get

Return Bounds?.TopLeft

End Get

Set(value As Point?)

Bounds = IfHasValue(value, AddressOf BoundsForNewPoint)

End Set

End Property

Protected MustOverride Function BoundsForNewPoint(point As Point) As Rectangle?

''' <returns>Whether the point was different from original.</returns>

Function Go(pointMap As Func(Of Point, Point)) As Boolean

Return IfHasValue(Position, Function(point)

Position = pointMap(point)

Return If(point <> Position, True)

End Function, False)

End Function

Public Overridable Function GoUp() As Boolean

Return Go(Function(point) New Point(point.Left, Math.Max(point.Top - 1, 0)))

End Function

Public Overridable Function GoDown() As Boolean

Return Go(Function(point) New Point(point.Left, Math.Min(WindowHeight - 1, point.Top + 1)))

End Function

Public Overridable Function GoLeft() As Boolean

Return Go(Function(point) New Point(Math.Max(point.Left - 1, 0), point.Top))

End Function

Public Overridable Function GoRight() As Boolean

Return Go(Function(point) New Point(Math.Min(point.Left + 1, WindowWidth - 2), point.Top)) ' Sunny is too fat and spans 2 spaces

End Function

Public Overridable Sub Dispose() Implements IDisposable.Dispose

Position = Nothing

End Sub

End Class

Class RectangleEntity

Inherits Entity

Public Sub New(entities As ICollection(Of Entity), rect As Rectangle?, Optional color As ConsoleColor = ConsoleColor.White)

MyBase.New(entities)

Me.Color = color

Rectangle = rect

End Sub

Protected Overrides Function ForbidEntry(other As Entity, otherBounds As Rectangle) As Boolean

Return IfHasValue(Bounds, Function(rect) \_

TypeOf other Is Entity AndAlso (

New Rectangle(rect.TopLeft, New Point(rect.Right, rect.Top)).SafeCollidesWith(otherBounds) OrElse

New Rectangle(rect.TopLeft, New Point(rect.Left, rect.Bottom)).SafeCollidesWith(otherBounds) OrElse

New Rectangle(New Point(rect.Left, rect.Bottom), New Point(rect.Right, rect.Bottom)).SafeCollidesWith(otherBounds) OrElse

New Rectangle(New Point(rect.Right, rect.Top), New Point(rect.Right, rect.Bottom)).SafeCollidesWith(otherBounds)), False)

End Function

Protected Overrides Function BoundsForNewPoint(point As Point) As Rectangle?

Return IfHasValue(Bounds, Function(rect) New Rectangle(point, rect.Width, rect.Height))

End Function

Public Property Color As ConsoleColor

Public Property Rectangle As Rectangle?

Get

Return Bounds

End Get

Set(value As Rectangle?)

Bounds = value

End Set

End Property

Protected Overrides Sub RedrawAt(bounds As Delta(Of Rectangle?))

If bounds.Changed Then

Dim Draw =

Sub(Rectangle As Rectangle?, horizontal As Char, vertical As Char,

topLeft As Char, topRight As Char, bottomLeft As Char, bottomRight As Char)

IfHasValue(Rectangle,

Sub(rect)

Dim DrawHorizontal =

Sub(y As Integer)

For x = 2 To rect.Width - 1 Step If(NeedDoubleRectangleWidth, 2, 1)

SetCursorPosition(rect.Left + x, y)

Write(horizontal)

Next

End Sub

ForegroundColor = Color

DrawHorizontal(rect.Bottom)

SetCursorPosition(rect.Left, rect.Bottom)

Write(bottomLeft)

SetCursorPosition(rect.Right, rect.Bottom)

Write(bottomRight)

For y = rect.Bottom - 1 To rect.Top + 1 Step -1

SetCursorPosition(rect.Left, y)

Write(vertical)

SetCursorPosition(rect.Right, y)

Write(vertical)

Next

DrawHorizontal(rect.Top)

CursorPosition = rect.TopLeft

Write(topLeft)

SetCursorPosition(rect.Right, rect.Top)

Write(topRight)

ResetColor()

End Sub)

End Sub

Draw(bounds.OldValue, Empty, Empty, Empty, Empty, Empty, Empty)

Draw(bounds.NewValue, "━"c, "┃"c, "┏"c, "┓"c, "┗"c, "┛"c)

End If

End Sub

End Class

Class TriggerZone

Inherits RectangleEntity

Public Sub New(entities As ICollection(Of Entity), rect As Rectangle?,

Optional keyPress As Func(Of ConsoleKey, Boolean) = Nothing,

Optional entityMove As Action = Nothing, Optional leave As Action = Nothing)

MyBase.New(entities, rect)

Me.EntityMove = entityMove

Me.Leave = leave

Me.KeyPress = keyPress

End Sub

Public Property EntityMove As Action

Dim EnterLock As Boolean

Public Property Leave As Action

Dim LeaveLock As Boolean

''' <returns>Whether the key has been handled.</returns>

Public Property KeyPress As Func(Of ConsoleKey, Boolean)

Protected Overrides Function ForbidEntry(other As Entity, otherNewBounds As Rectangle) As Boolean

If (TypeOf other Is PlayerEntity) Then

Dim player = DirectCast(other, PlayerEntity)

If Bounds?.PreciseCollidesWith(otherNewBounds) Then

player.Trigger = Me

If Not EnterLock Then

EnterLock = True

EntityMove?.Invoke()

EnterLock = False

End If

ElseIf player.Trigger Is Me Then

If Not LeaveLock Then

LeaveLock = True

player.Trigger = Nothing

Leave?.Invoke()

LeaveLock = False

End If

End If

End If

Return False

End Function

Public Overrides Sub Dispose()

If ActiveEntity.Trigger Is Me Then

ActiveEntity.Trigger = Nothing

Leave?.Invoke()

End If

MyBase.Dispose()

End Sub

Protected Overrides Sub RedrawAt(bounds As Delta(Of Rectangle?)) ' Doesn't need to be drawn

End Sub

End Class

Class TextEntity

Inherits Entity

Public Sub New(entities As ICollection(Of Entity), text As String,

Optional position As Point? = Nothing, Optional color As ConsoleColor = ConsoleColor.White)

MyBase.New(entities)

Me.Color = color

Me.Text = text

Me.Position = position

End Sub

Protected Overrides Function BoundsForNewPoint(point As Point) As Rectangle?

Return New Rectangle(point, Text.Length, 1)

End Function

Dim \_text As String

Public Property Text As String

Get

Return \_text

End Get

Set(value As String)

RedrawAt(New Delta(Of Rectangle?)(Bounds), New Delta(Of String)(\_text, value))

\_text = value

End Set

End Property

Public Property Color As ConsoleColor

Protected Overrides Sub RedrawAt(bounds As Delta(Of Rectangle?))

RedrawAt(bounds, New Delta(Of String)(\_text))

End Sub

Protected Overloads Sub RedrawAt(bounds As Delta(Of Rectangle?), text As Delta(Of String))

IfHasValue(bounds.OldValue, Sub(point)

ForegroundColor = Color

CursorPosition = point.TopLeft

For i = 1 To text.OldValue.Length

Write(Empty)

Next

End Sub)

IfHasValue(bounds.NewValue, Sub(point)

ForegroundColor = Color

CursorPosition = point.TopLeft

Write(text.NewValue)

End Sub)

End Sub

End Class

Class SpriteEntity

Inherits Entity

Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)

MyBase.New(entities)

\_sprite = sprite

End Sub

Protected Overrides Function BoundsForNewPoint(point As Point) As Rectangle?

Return New Rectangle(point, 1, 1)

End Function

Dim \_sprite As Sprite

Public Property Sprite As Sprite

Get

Return \_sprite

End Get

Set(value As Sprite)

RedrawAt(New Delta(Of Rectangle?)(Bounds), New Delta(Of Sprite)(\_sprite, value))

\_sprite = value

End Set

End Property

Protected Overrides Sub RedrawAt(bounds As Delta(Of Rectangle?))

RedrawAt(bounds, New Delta(Of Sprite)(\_sprite))

End Sub

Protected Overloads Sub RedrawAt(bounds As Delta(Of Rectangle?), sprite As Delta(Of Sprite))

If bounds.Changed Then WriteAt(bounds.OldValue?.TopLeft, Empty\_)

WriteAt(bounds.NewValue?.TopLeft, sprite.NewValue)

End Sub

End Class

MustInherit Class TickEntity

Inherits SpriteEntity

Protected MustOverride Sub OnTick()

Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)

MyBase.New(entities, sprite)

AddHandler Tick, AddressOf OnTick

End Sub

Public Overrides Sub Dispose()

RemoveHandler Tick, AddressOf OnTick

MyBase.Dispose()

End Sub

End Class

Class IteratingSpriteEntity

Inherits TickEntity

ReadOnly sprites As IEnumerator(Of Sprite)

Public Sub New(entities As ICollection(Of Entity), sprites As IEnumerable(Of Sprite), position As Point)

MyBase.New(entities, sprites.First())

Me.Position = position

Me.sprites = sprites.Skip(1).GetEnumerator()

End Sub

Protected Overrides Sub OnTick()

If sprites.MoveNext() Then Sprite = sprites.Current Else Dispose()

End Sub

Public Overrides Sub Dispose()

sprites.Dispose()

MyBase.Dispose()

End Sub

End Class

Class GravityEntity

Inherits TickEntity

Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)

MyBase.New(entities, sprite)

End Sub

Protected Overrides Sub OnTick()

If VerticalVelocity > 0 Then

MyBase.GoUp()

VerticalVelocity -= 1

ElseIf MyBase.GoDown() Then

VerticalVelocity -= 1

Else

VerticalVelocity = 0

RaiseEvent GroundHit()

GroundHitEvent = Nothing

End If

End Sub

Public Property VerticalVelocity As Integer

Public Event GroundHit()

Public Overrides Function GoUp() As Boolean

If MyBase.GoDown() Then

Return False ' Can't jump while falling

Else

MyBase.GoUp()

VerticalVelocity = 2

Return True

End If

End Function

End Class

Class PlayerEntity

Inherits GravityEntity

Public Property Trigger As TriggerZone

Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)

MyBase.New(entities, sprite)

End Sub

Public Sub HandleKey(key As ConsoleKey)

If Trigger?.KeyPress Is Nothing OrElse Not Trigger.KeyPress(key) Then

Select Case key

Case ConsoleKey.LeftArrow : GoLeft()

Case ConsoleKey.RightArrow : GoRight()

Case ConsoleKey.UpArrow : GoUp()

Case ConsoleKey.DownArrow : GoDown()

End Select

End If

End Sub

Public Overrides Function GoLeft() As Boolean

Dim ret = MyBase.GoLeft()

If Position?.Left = 0 AndAlso CurrentRegion.GoLeft() Then

Position = New Point(WindowWidth - 3, Position.GetValueOrDefault().Top)

End If

Return ret

End Function

Public Overrides Function GoRight() As Boolean

Dim ret = MyBase.GoRight()

If Position?.Left = WindowWidth - 2 AndAlso CurrentRegion.GoRight() Then

Position = New Point(1, Position.GetValueOrDefault().Top)

End If

Return ret

End Function

End Class

Class GravityEntityFactory

Class GravityEntityFactoryEntity

Inherits GravityEntity

Friend Owner As GravityEntityFactory

Public Sub New(entities As ICollection(Of Entity), sprite As Sprite,

owner As GravityEntityFactory, position As Point?, onHitGround As GroundHitEventHandler)

MyBase.New(entities, sprite)

Me.Owner = owner

Me.Position = position

If onHitGround IsNot Nothing Then AddHandler GroundHit, onHitGround

End Sub

End Class

Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)

Me.Entities = entities

Me.Sprite = sprite

End Sub

ReadOnly Sprite As Sprite

ReadOnly Entities As ICollection(Of Entity)

Public Property Template As Sprite

Public Sub Add(position As Point?, Optional onHitGround As GravityEntity.GroundHitEventHandler = Nothing)

Entities.Add(New GravityEntityFactoryEntity(Entities, Sprite, Me, position, onHitGround) With {.VerticalVelocity = 1})

End Sub

Public Sub Clear()

For Each item In Entities.OfType(Of GravityEntityFactoryEntity).Where(Function(e) e.Owner Is Me)

Entities.Remove(item)

item.Dispose()

Next

End Sub

Public Function ItemAt(position As Point) As GravityEntity

For Each item In Entities.OfType(Of GravityEntityFactoryEntity).Where(Function(e) e.Owner Is Me)

If item.Position = position Then Return item

Next

Return Nothing

End Function

End Class

The implementation of each non-game processes are best described through the explanation of the main function which is the entry point. The function and characteristics of the main function will be mentioned one by one in this section.



**3.6: Process implementation (Games)**

Implementation of each the game processes are contained within “regions” which act as a room for the avatar called Sunny to travel through. The implementation of each game processes are best described through the explanation of every region. The function and characteristics of every region will be mentioned one by one in this section.

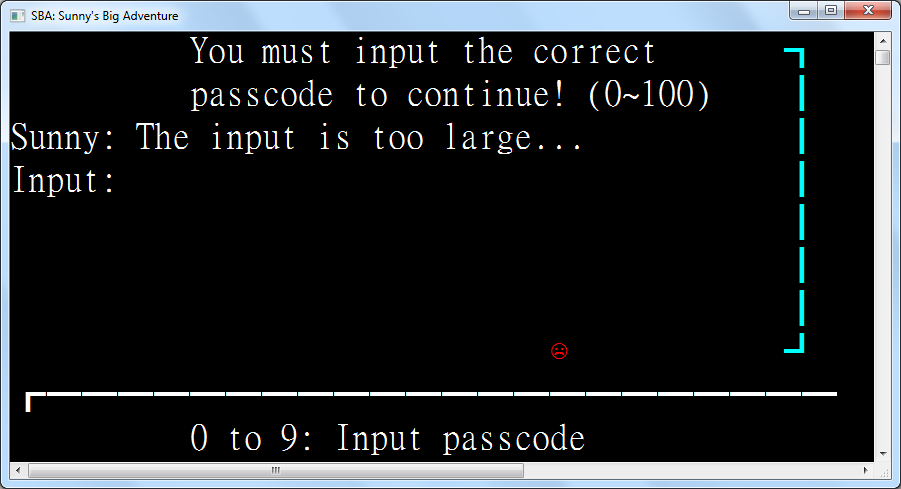
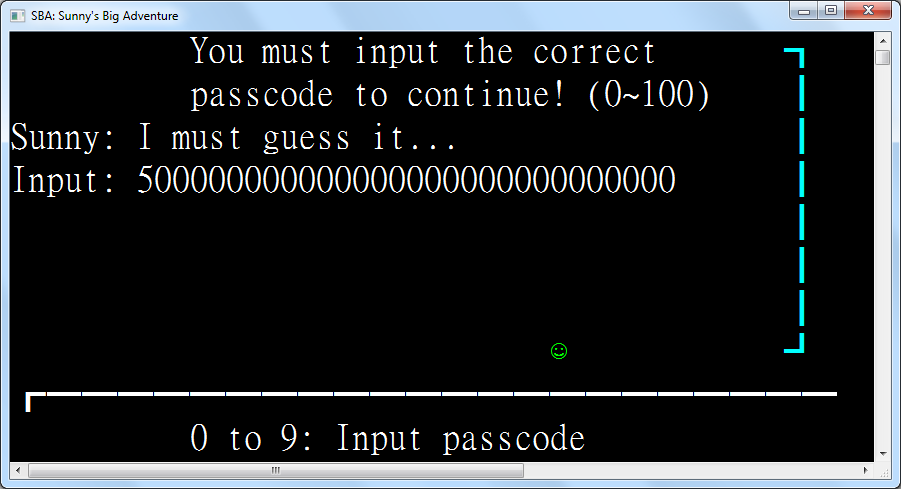
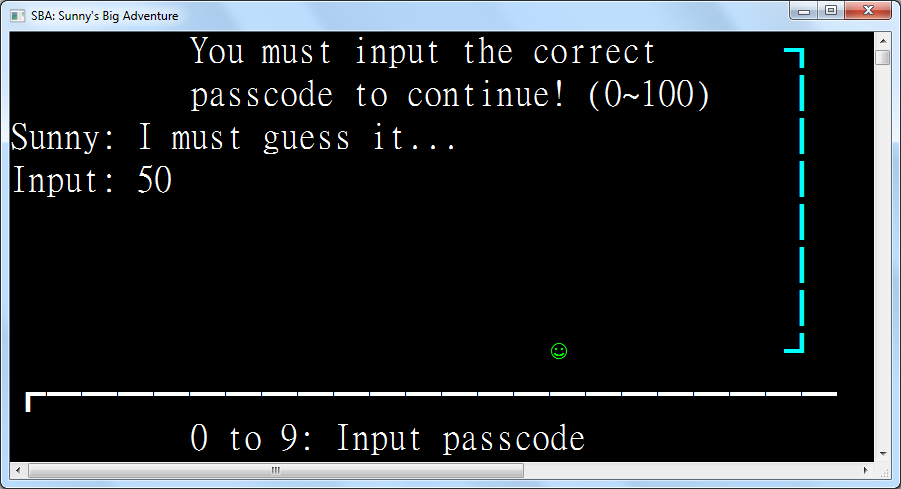
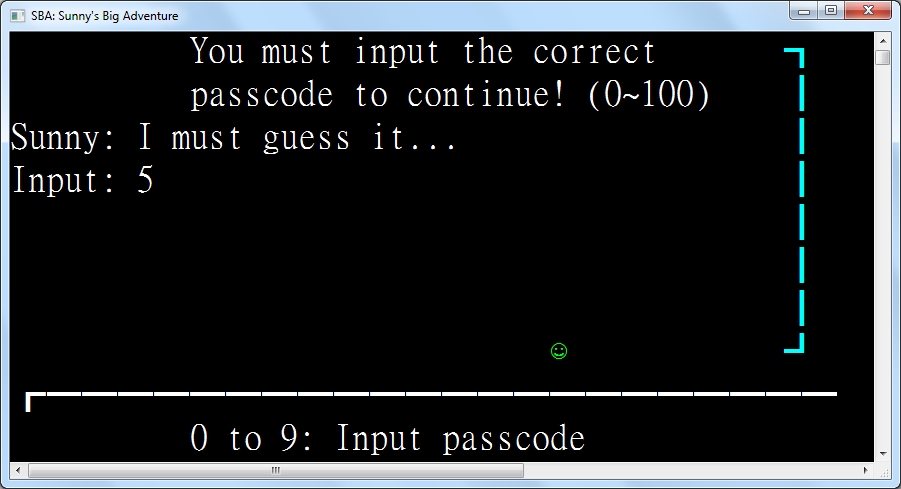
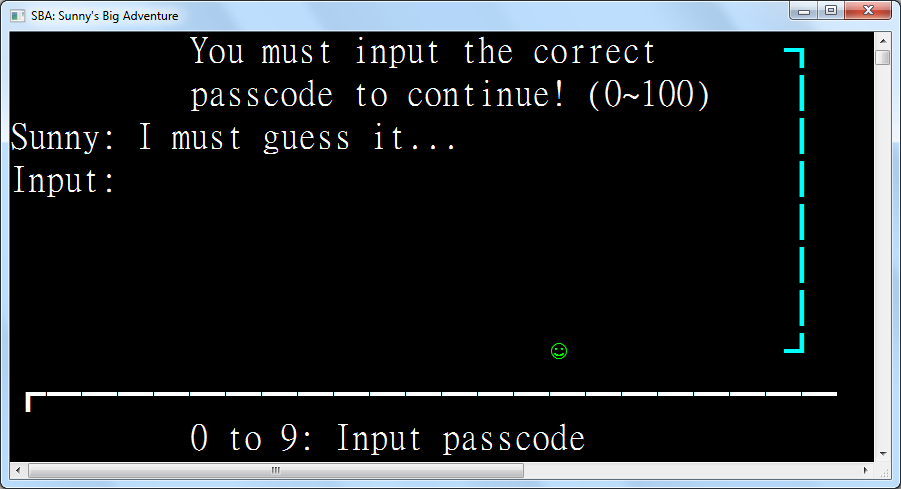
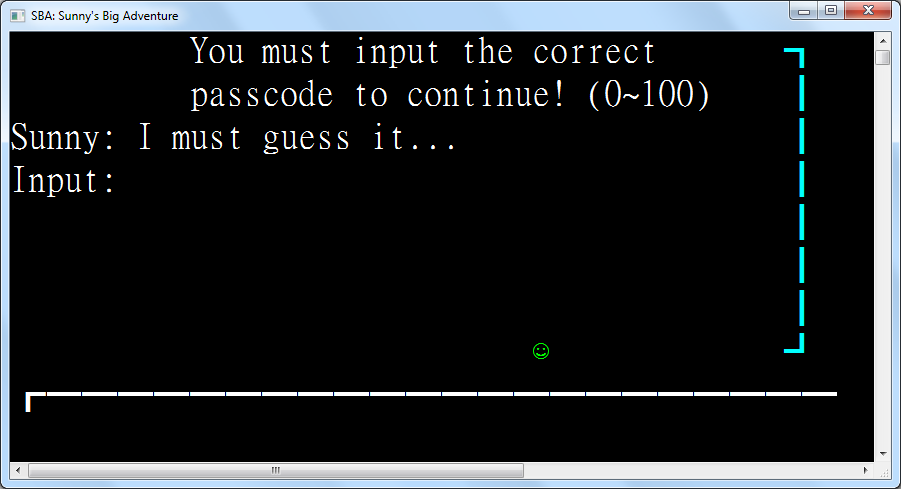
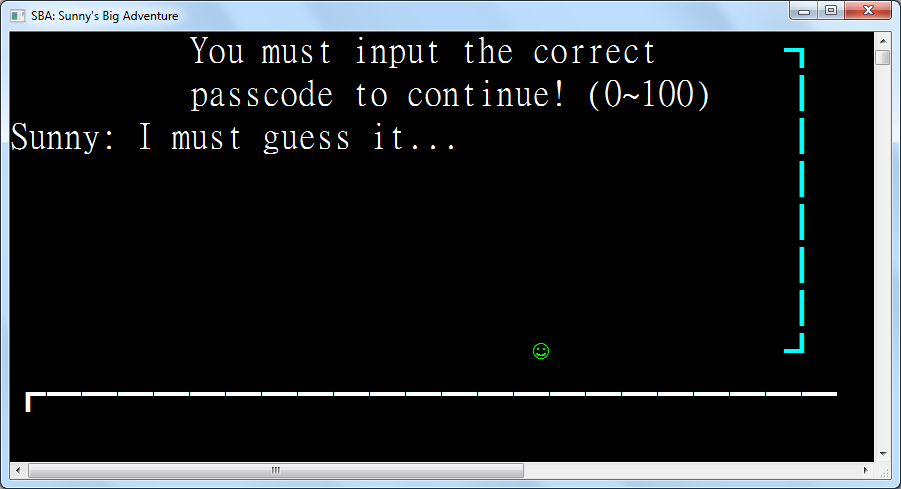
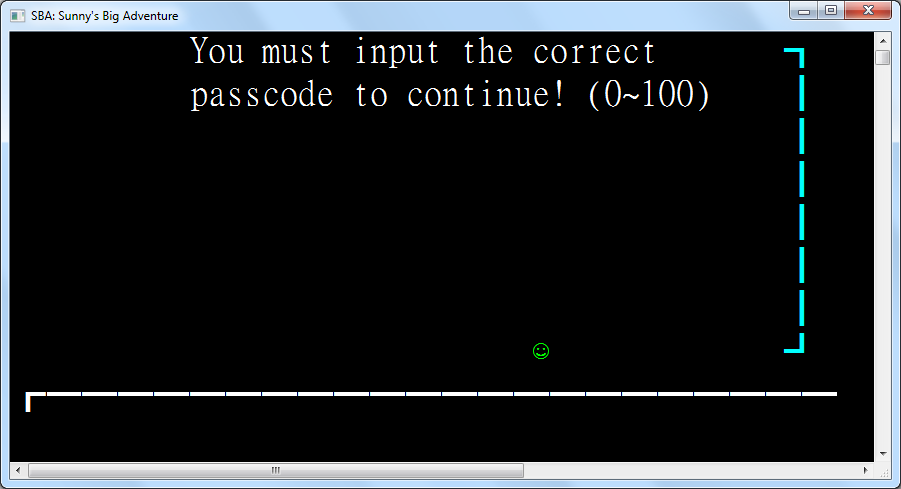
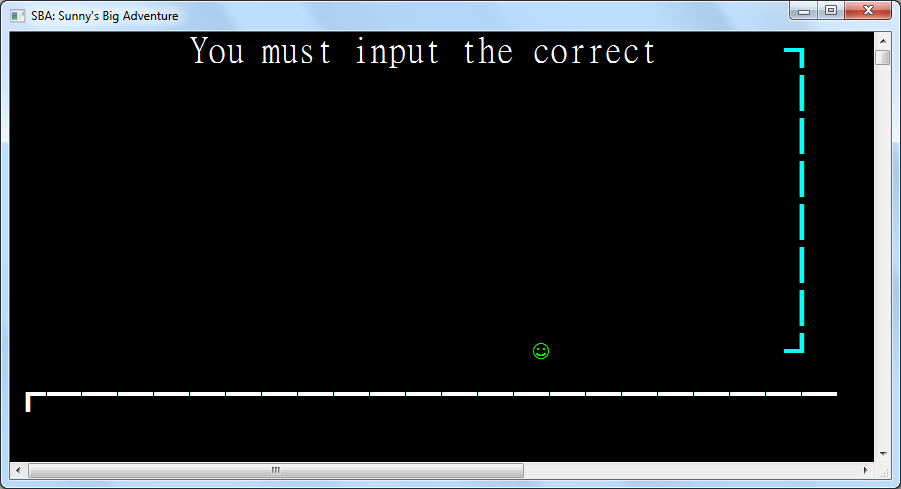
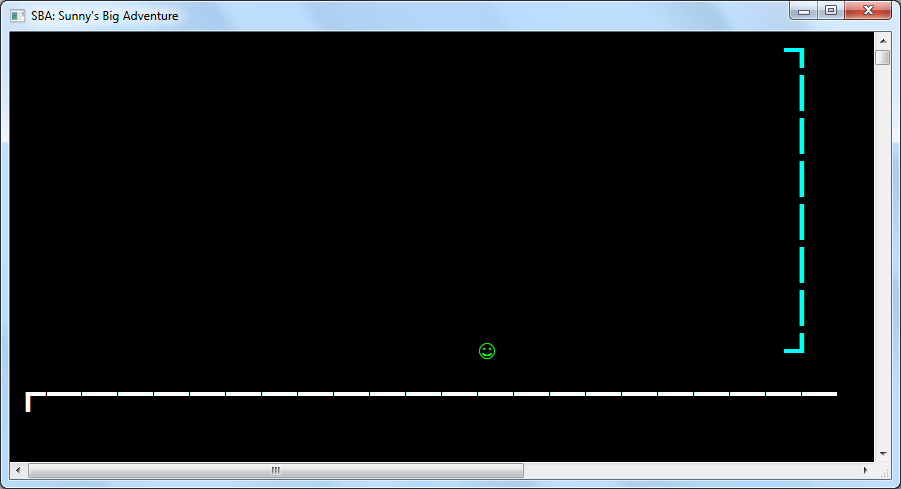
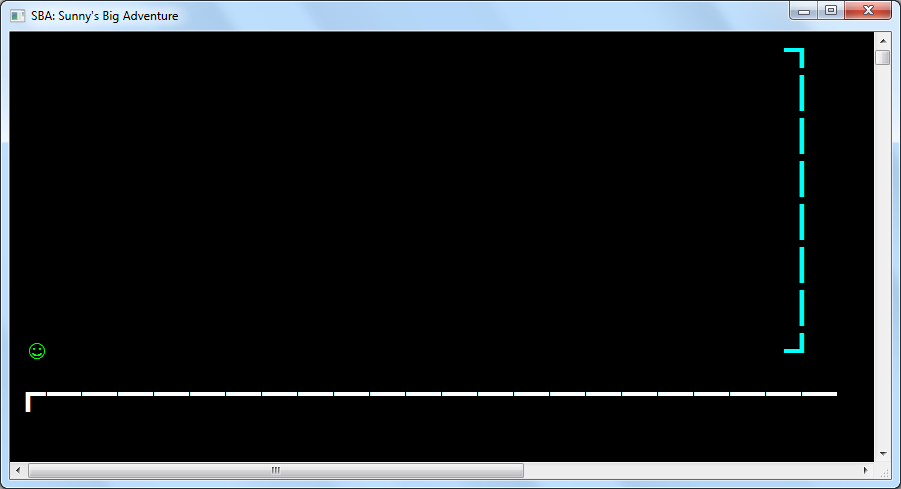
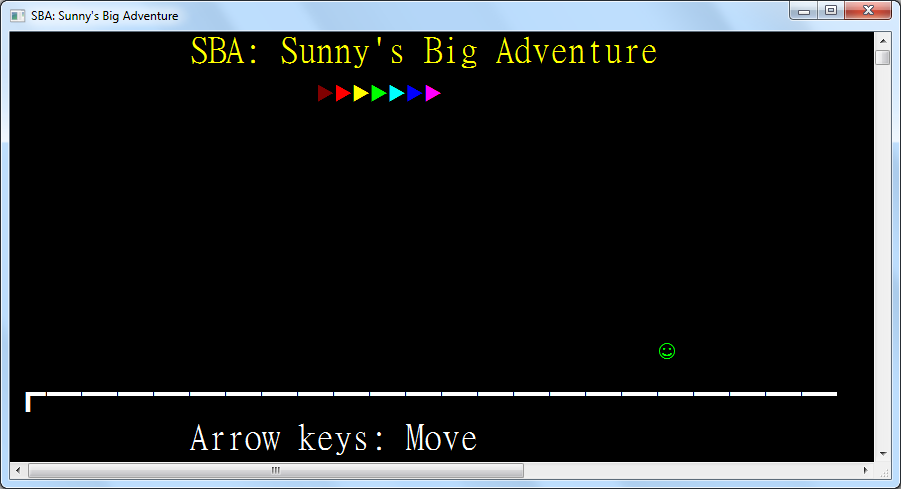
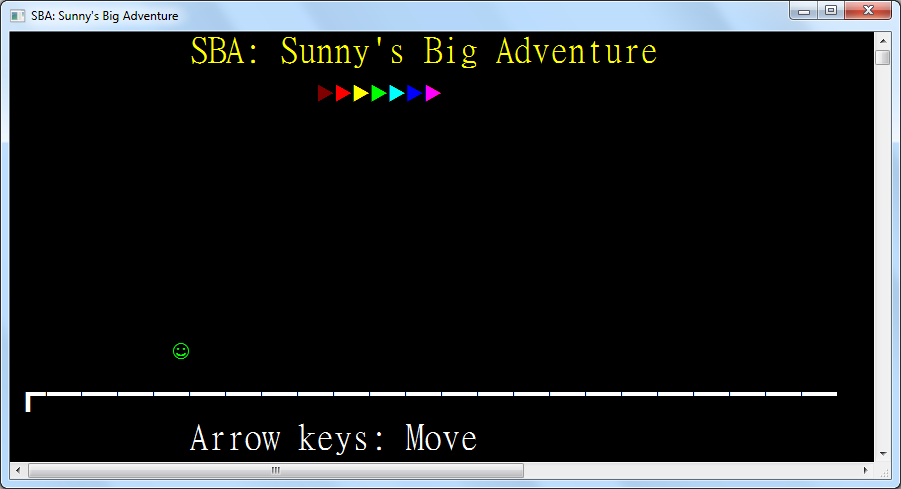
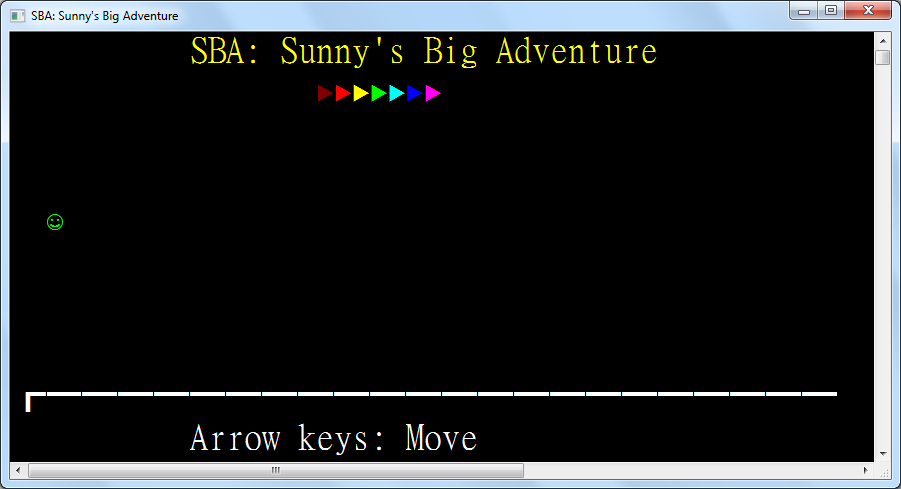
**3.6.1: Region 1 – Title Screen**

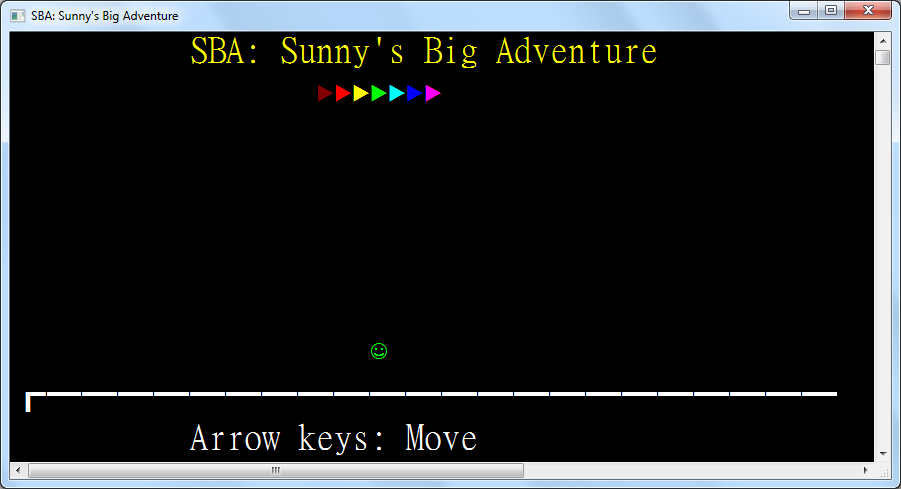
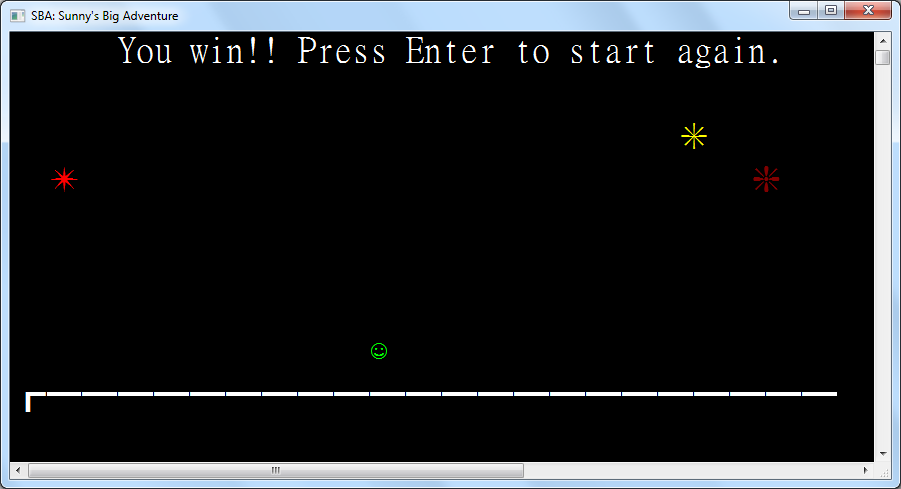
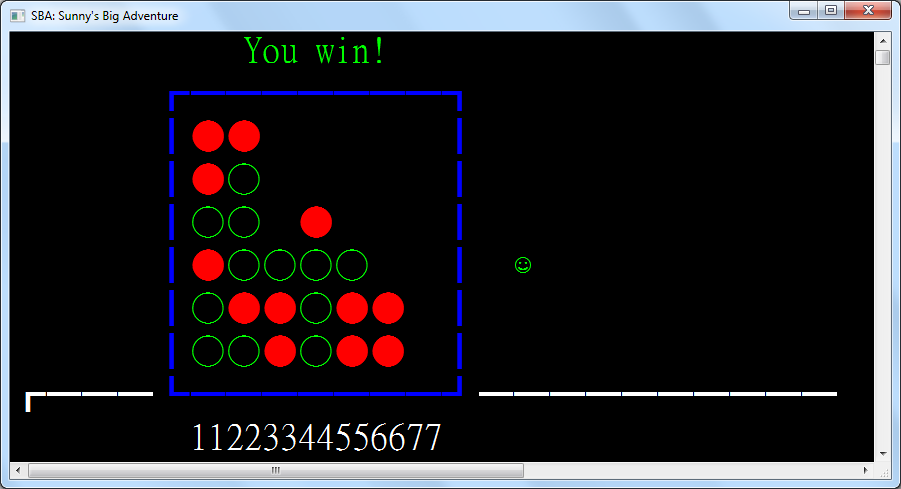
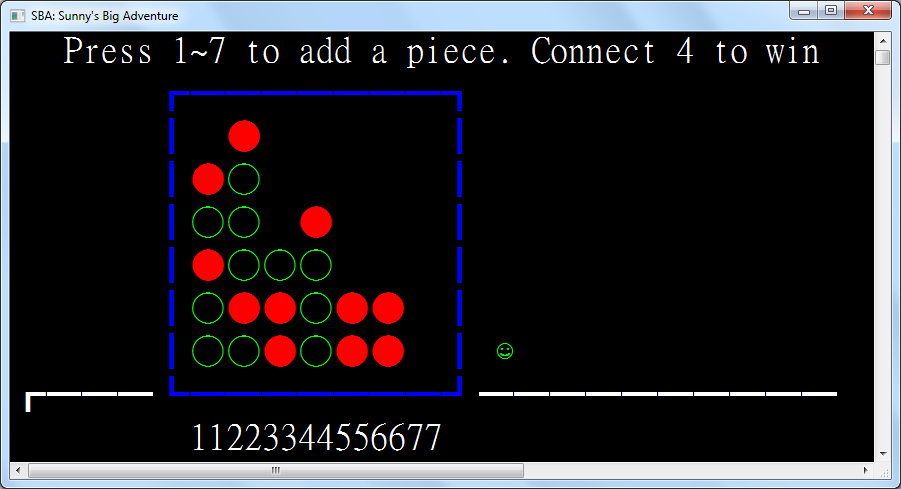
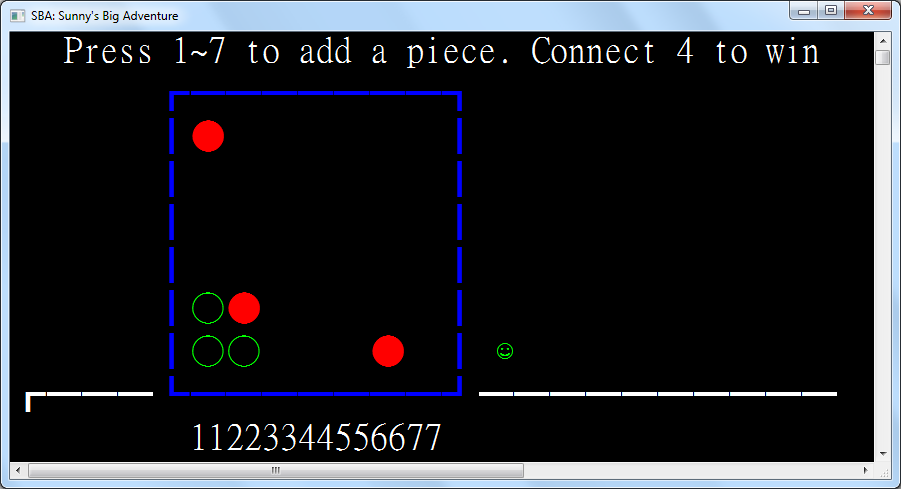
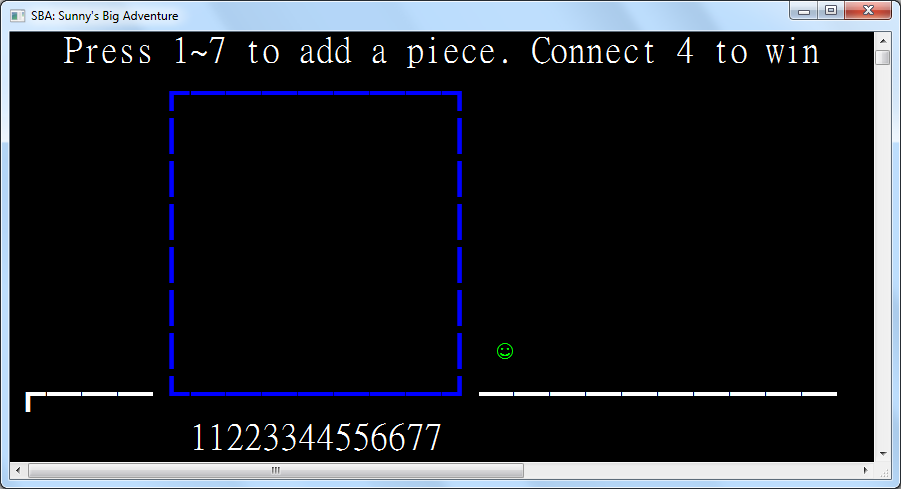
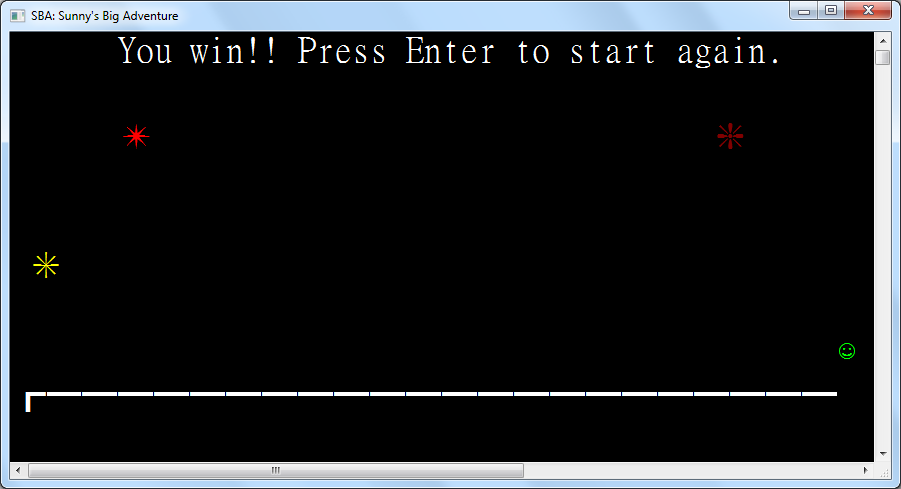
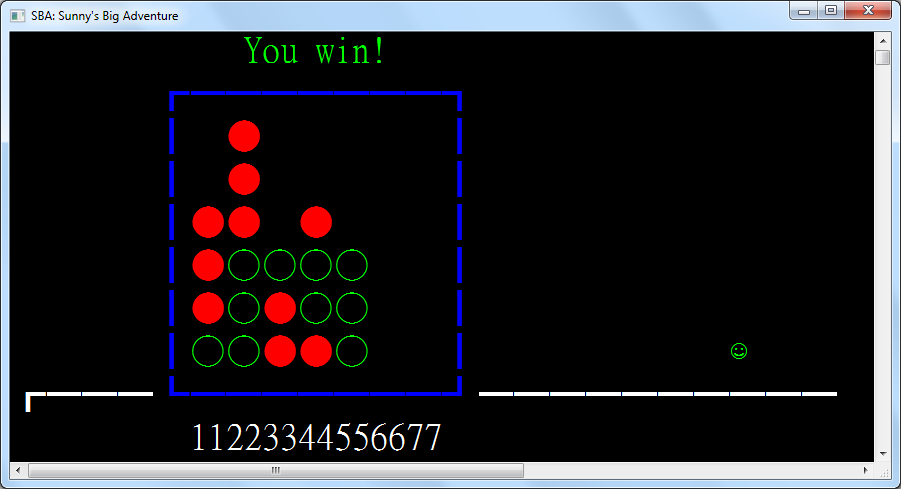
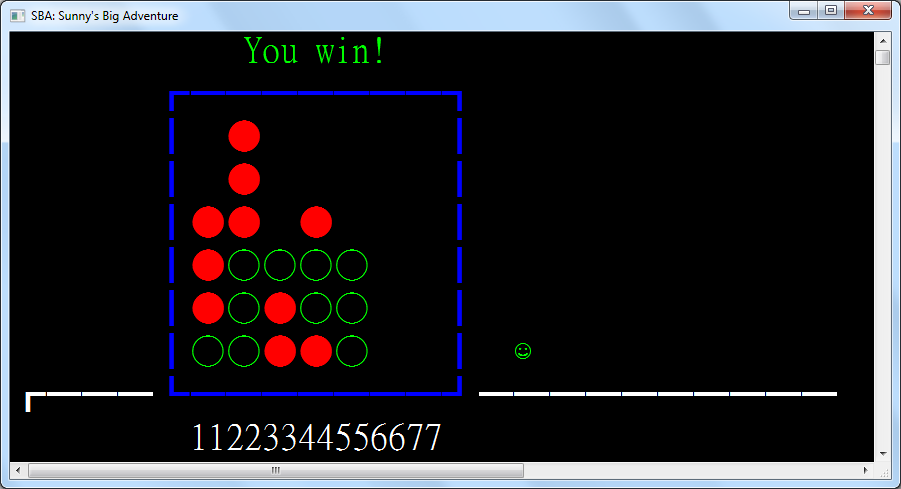
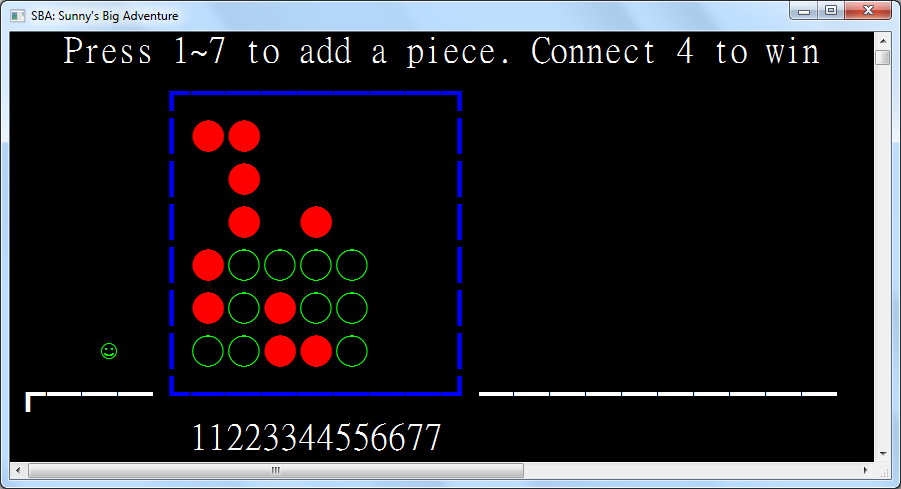
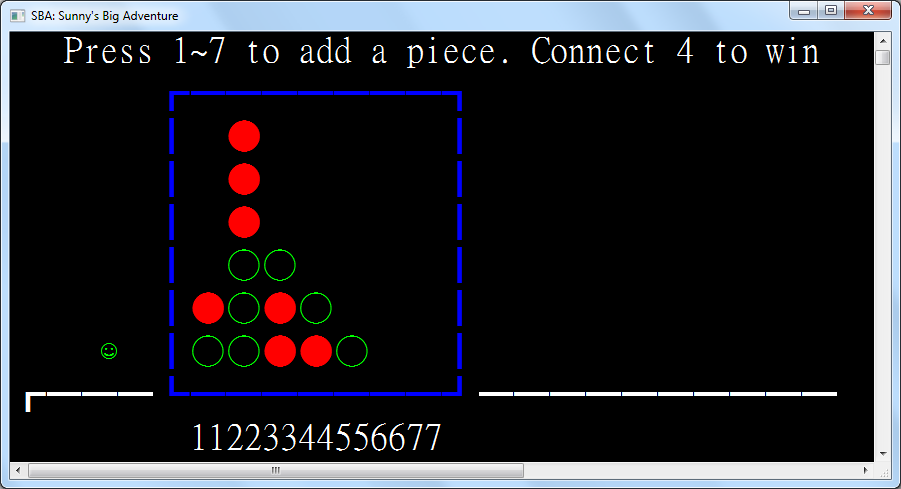
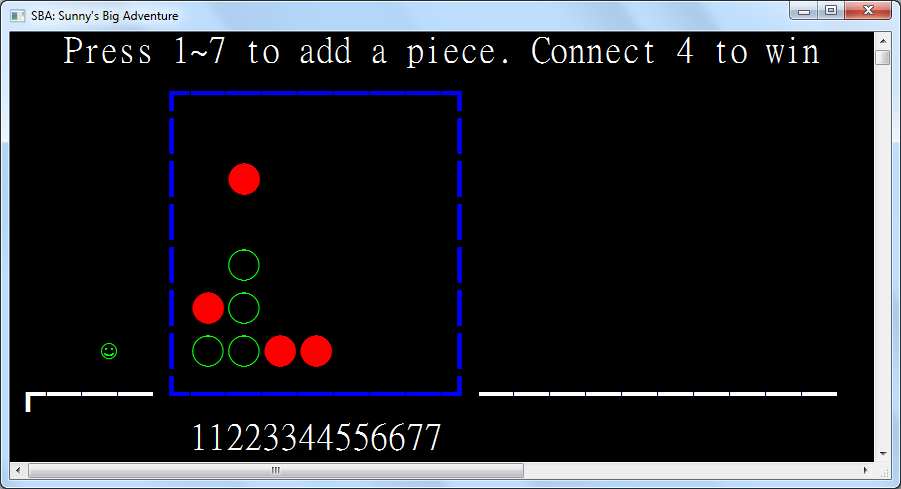
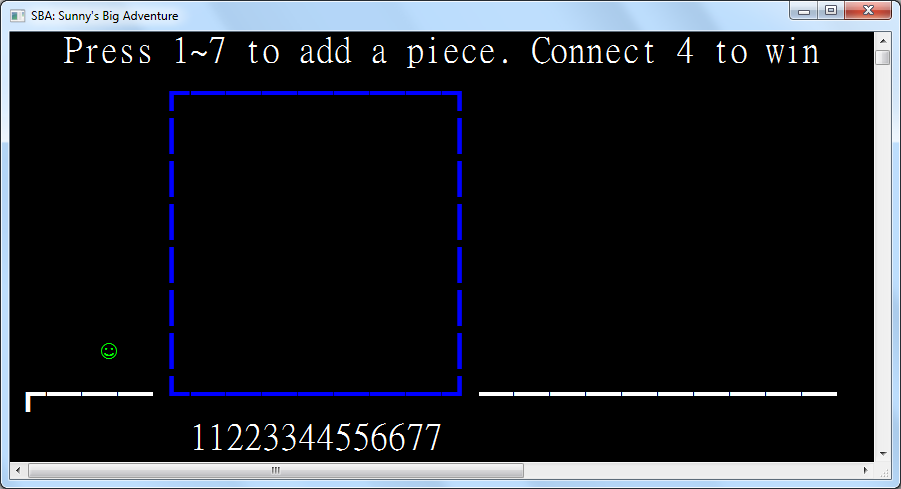
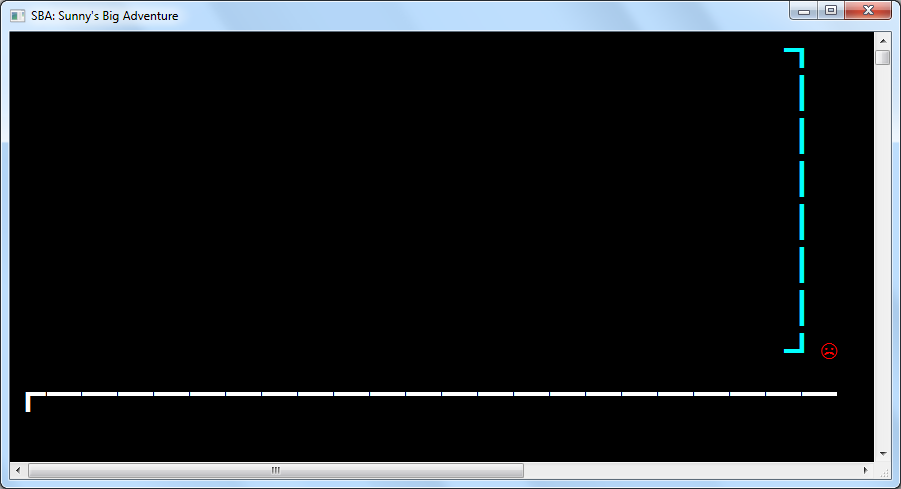
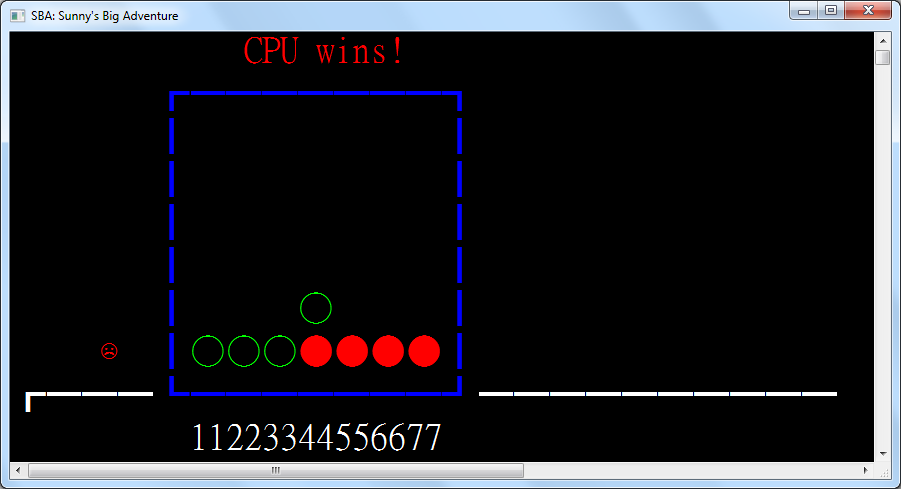
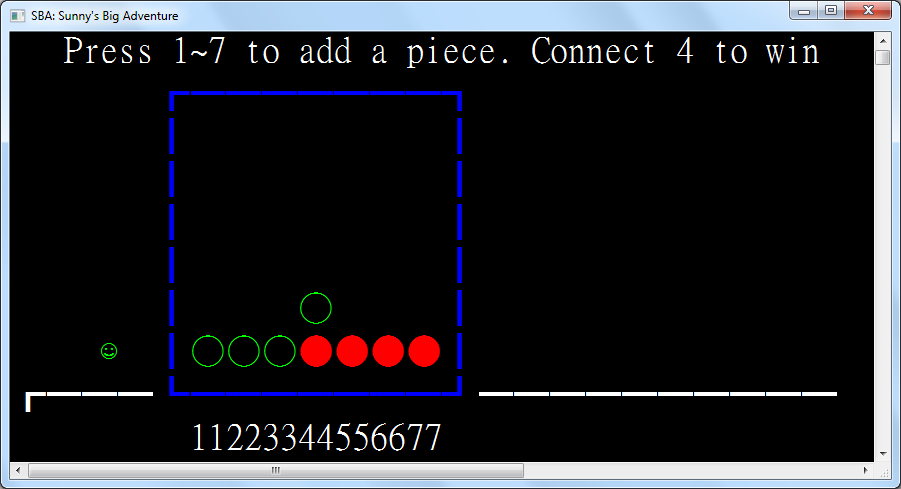
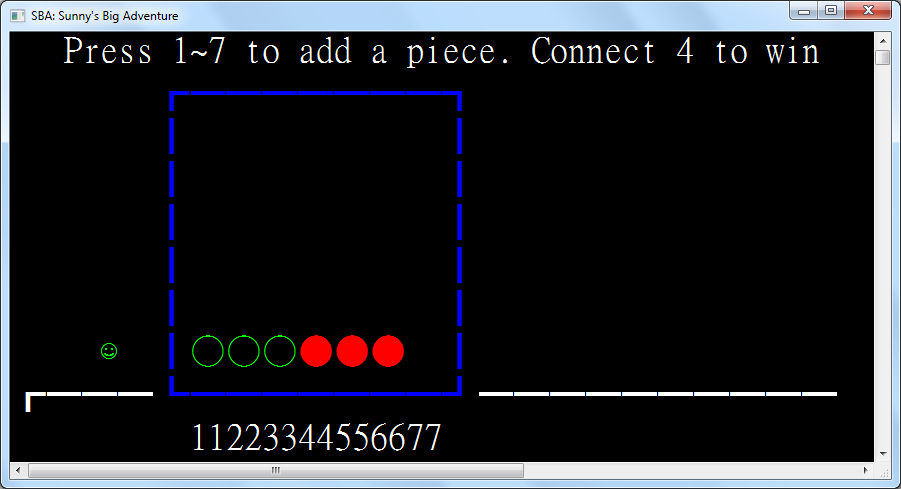
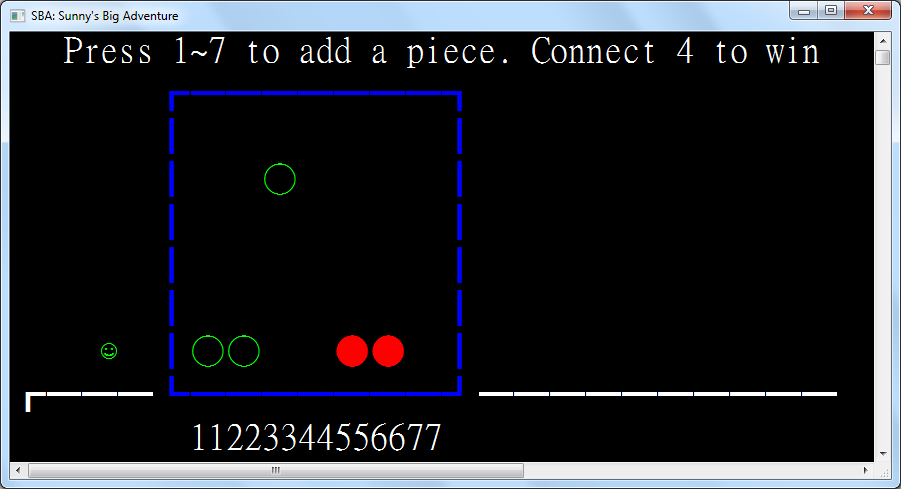
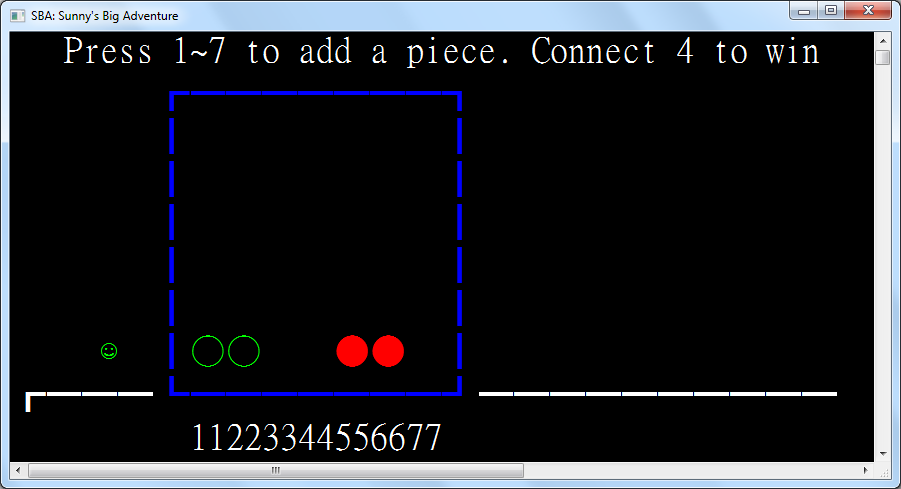
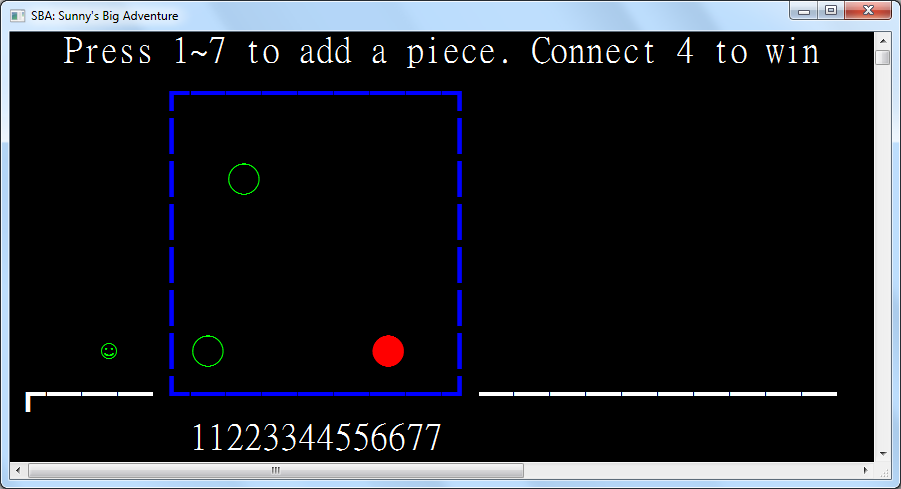
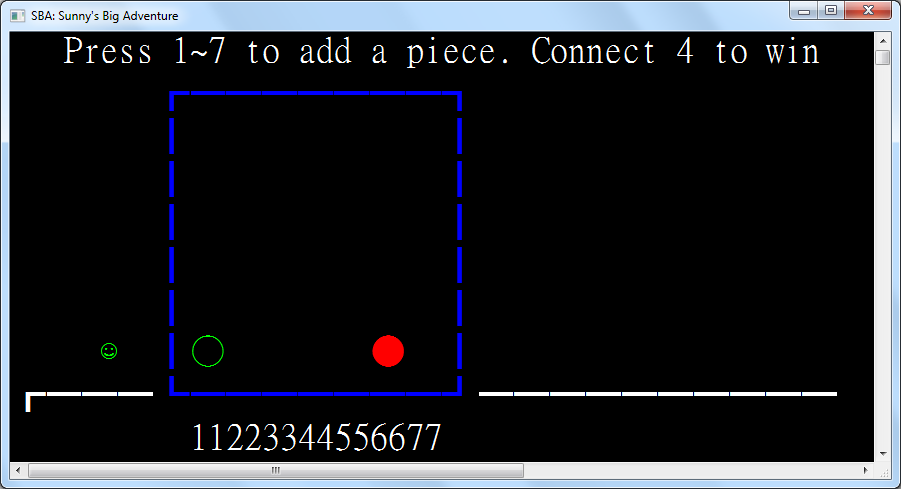
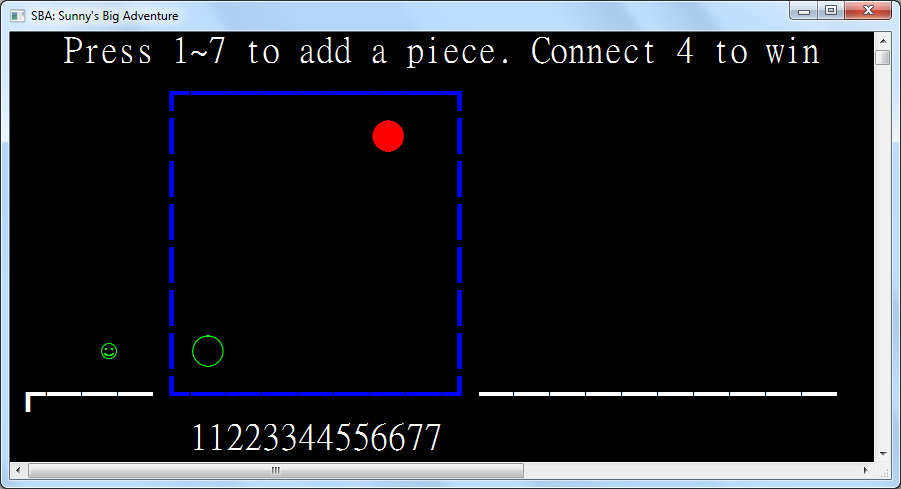
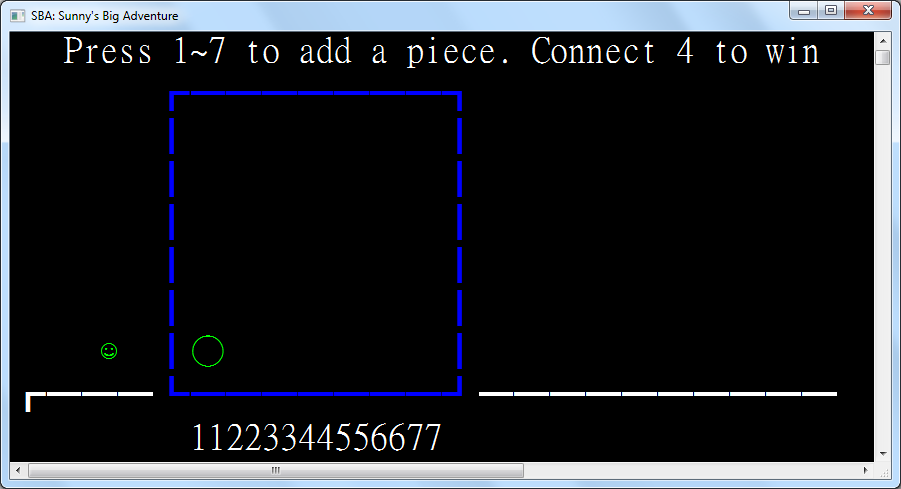
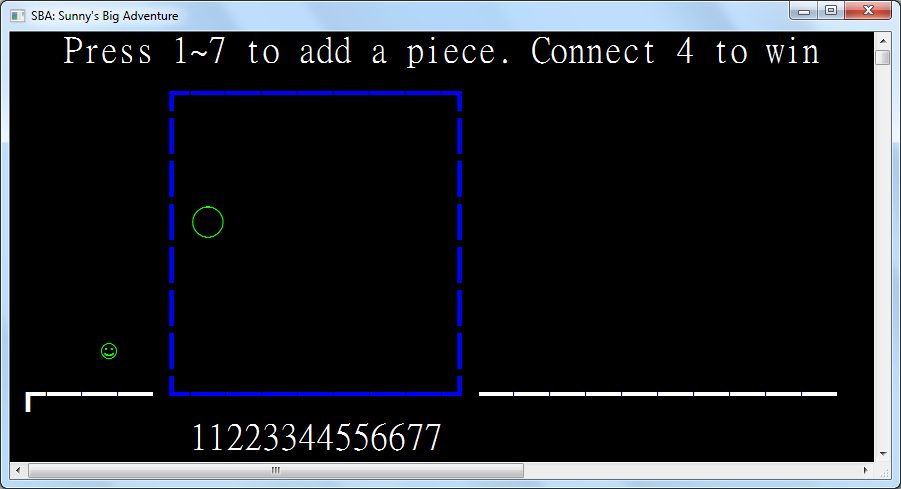
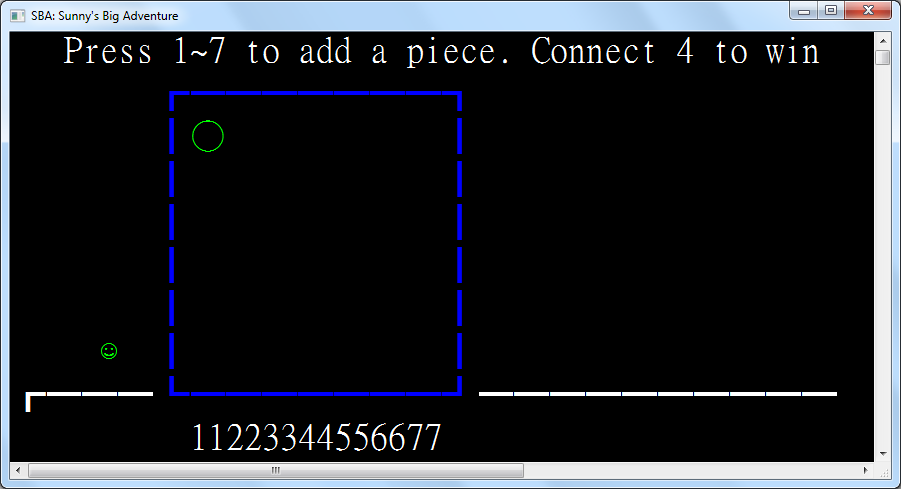
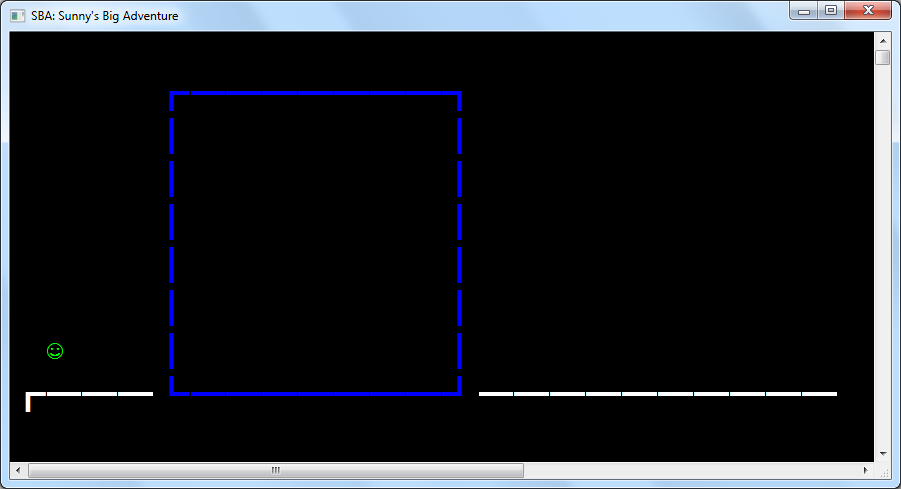
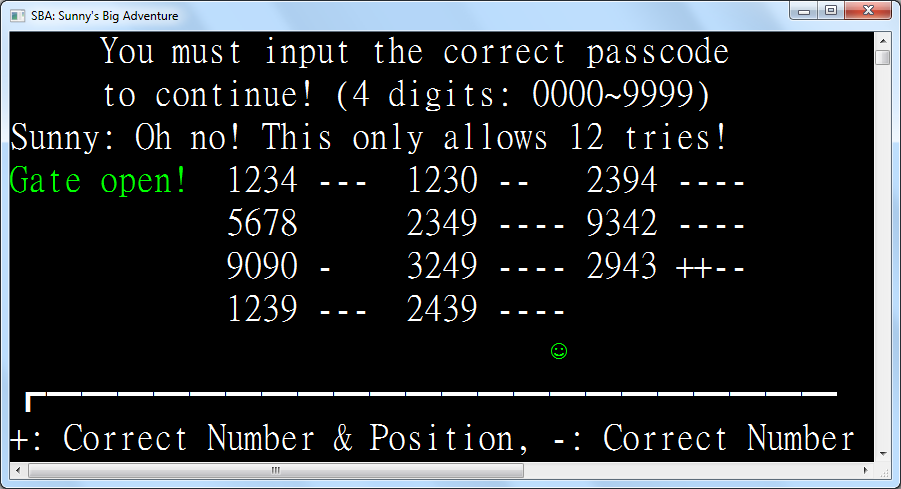
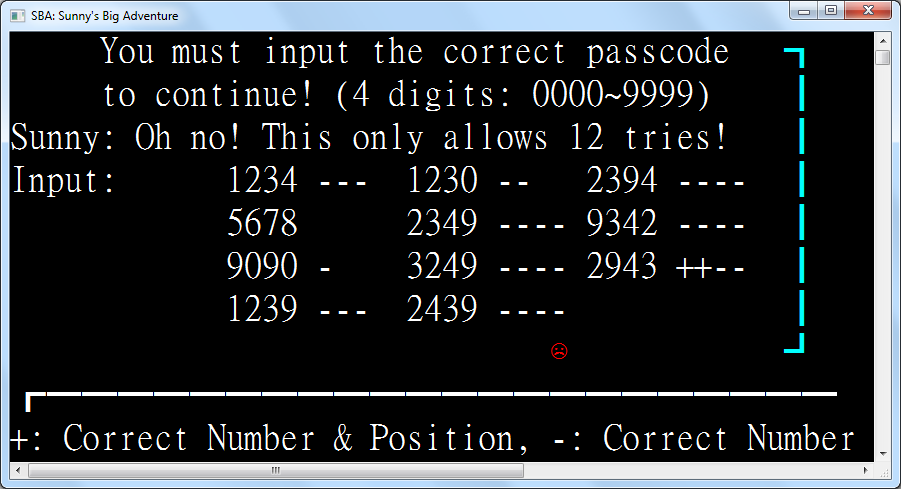
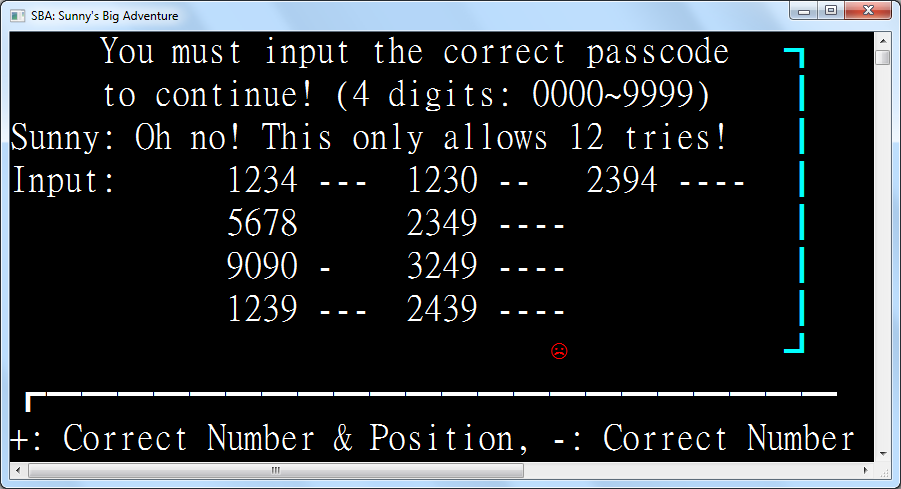
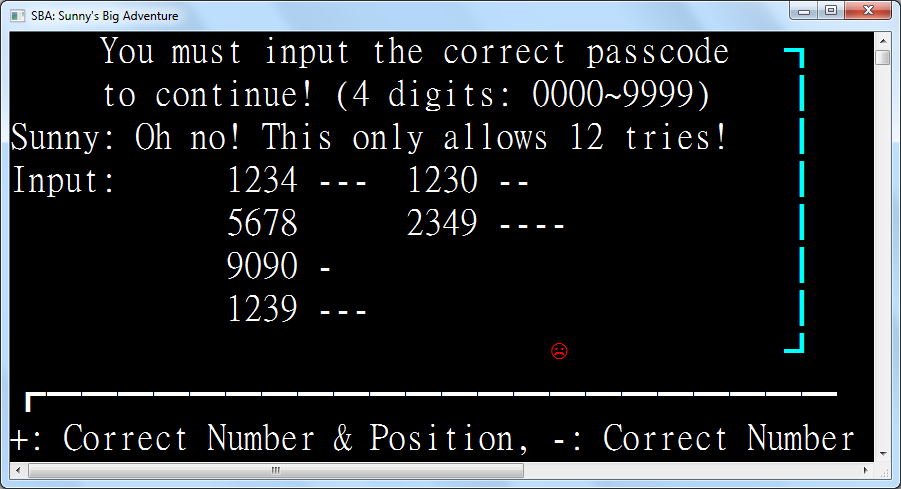
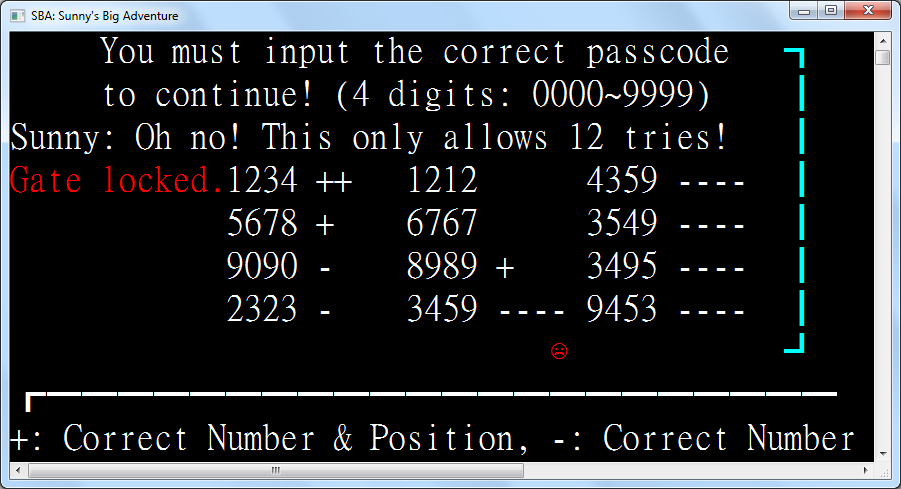
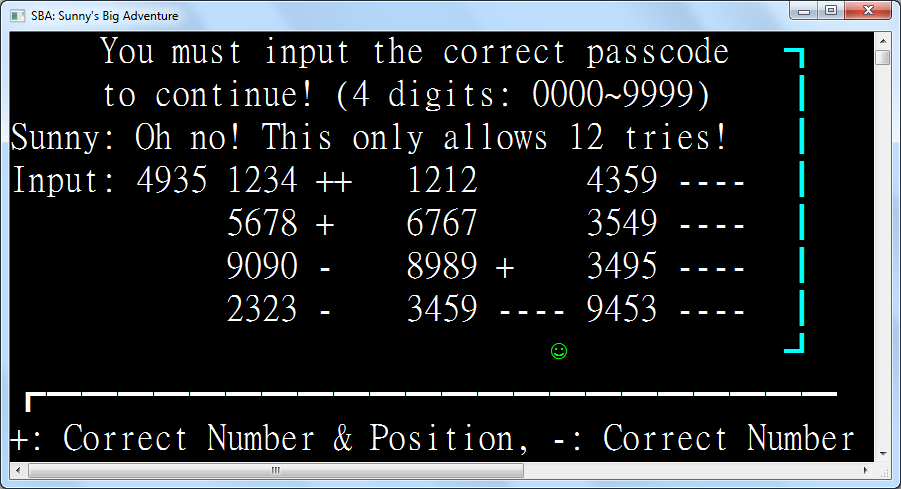
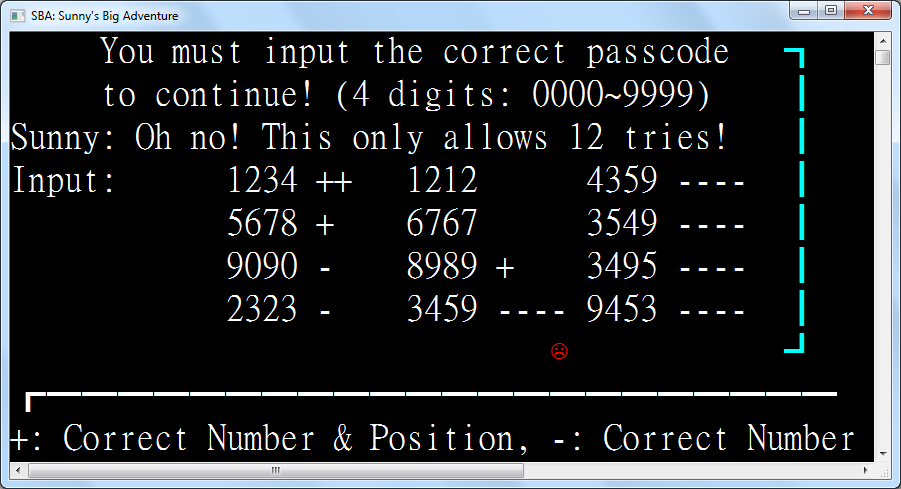
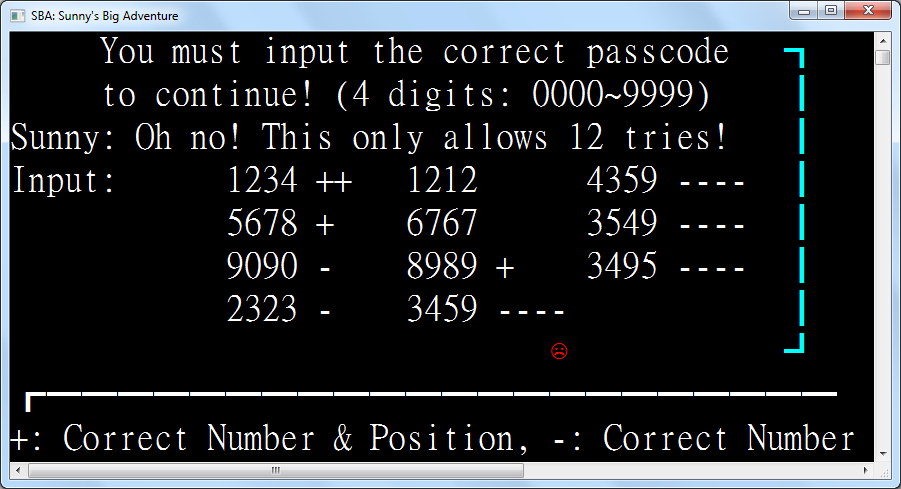
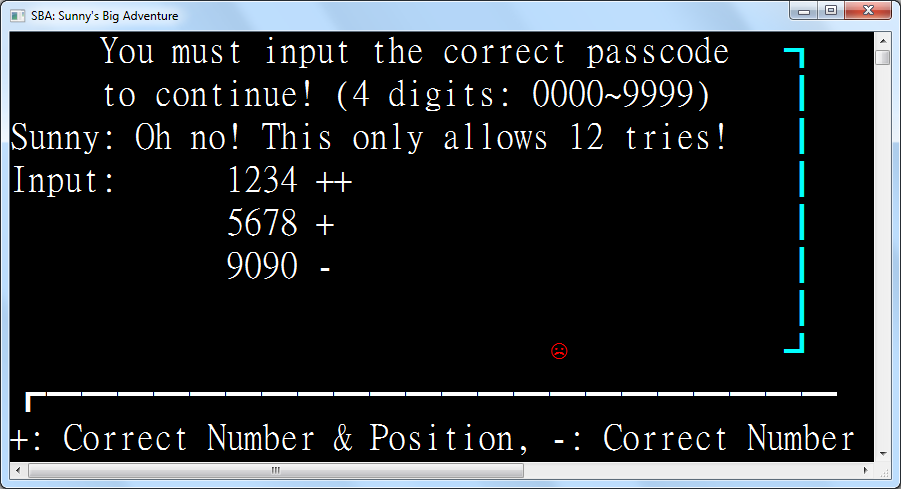
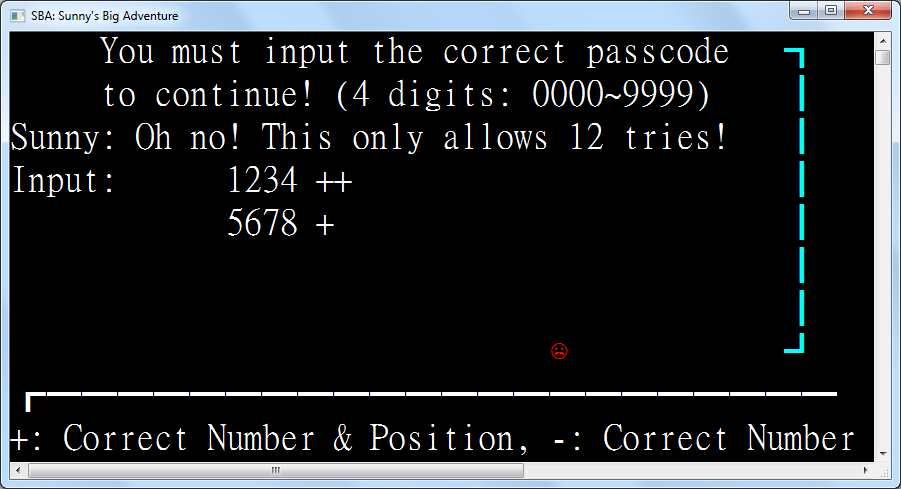
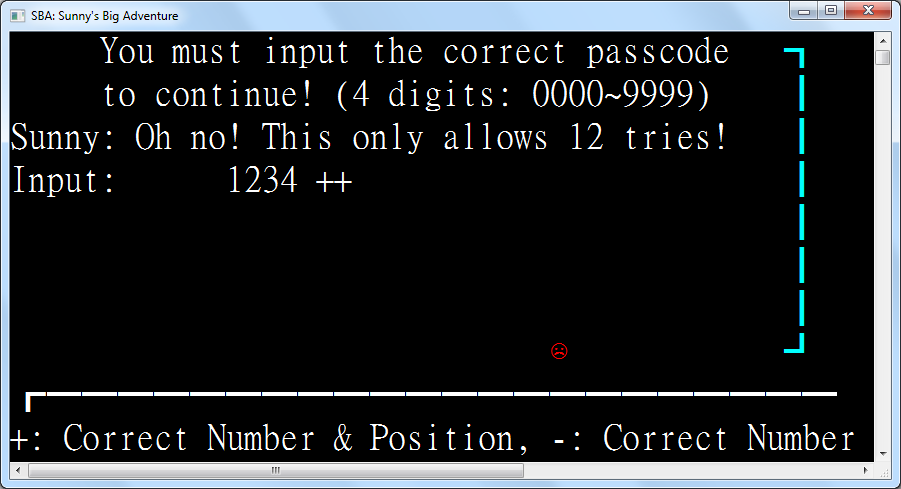
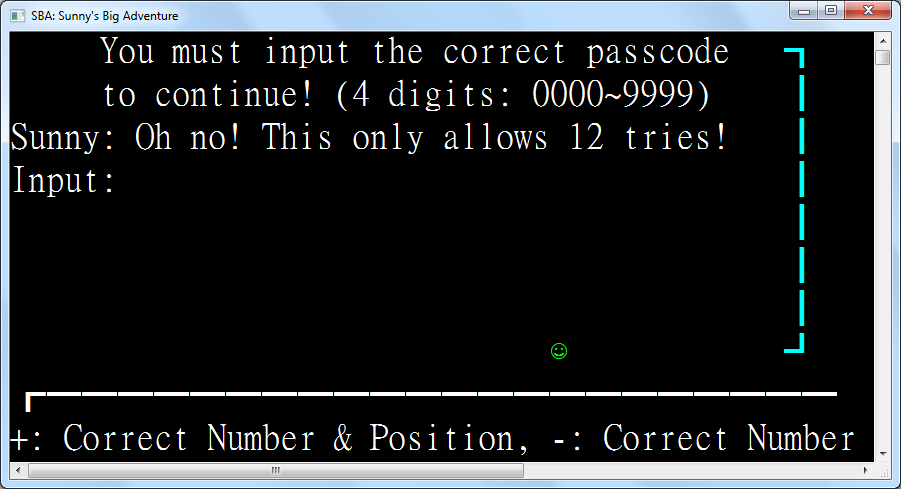
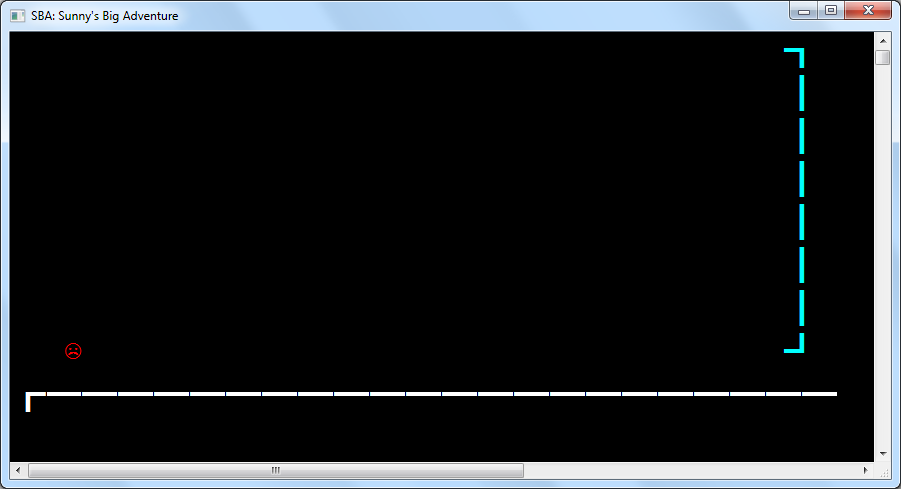
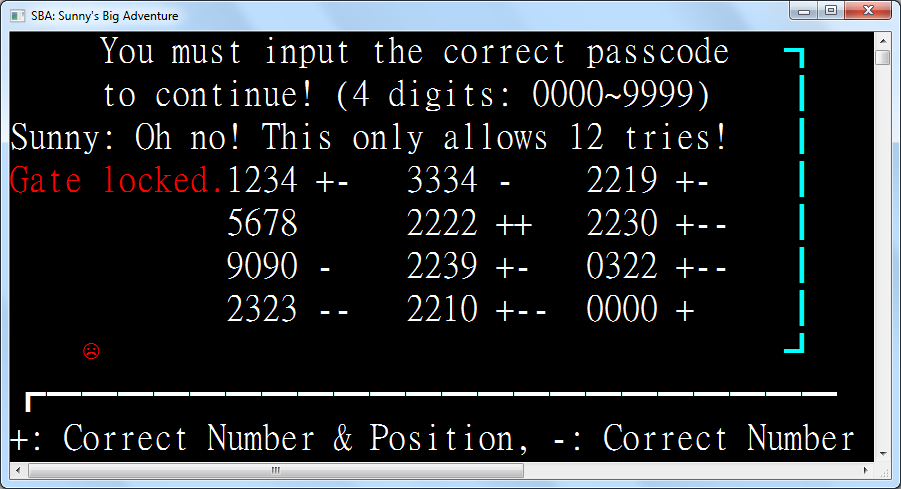
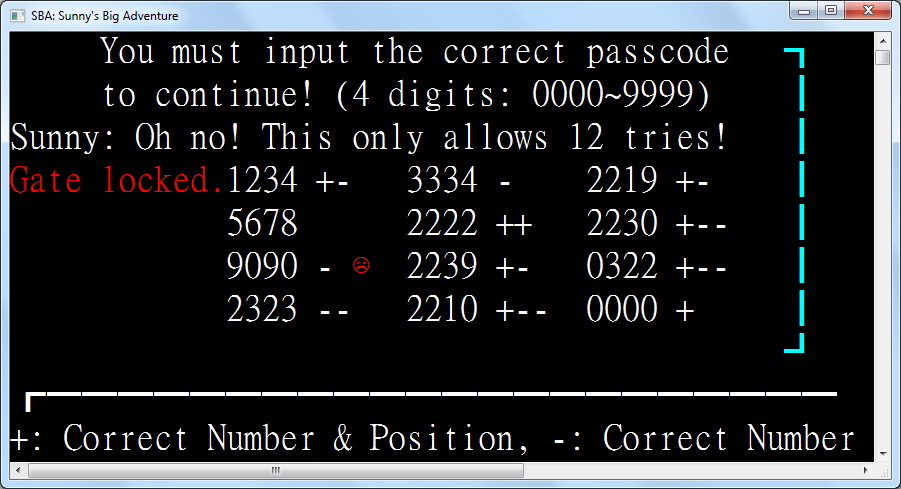
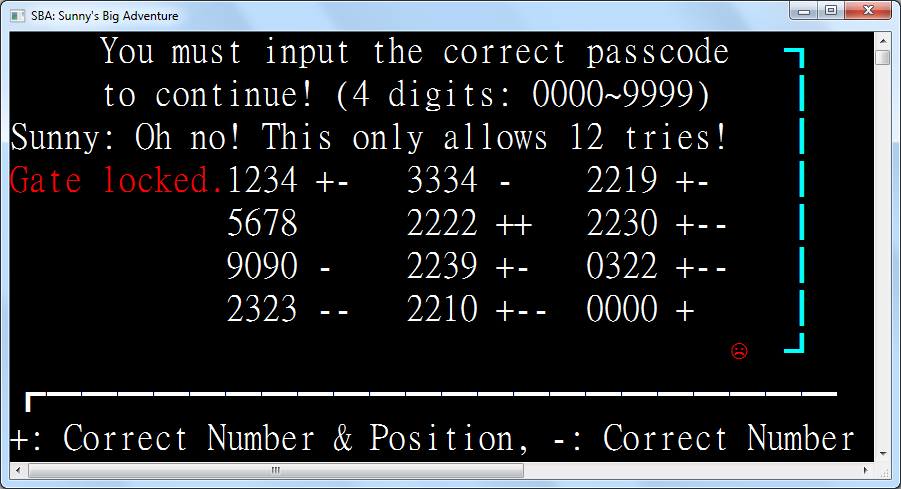
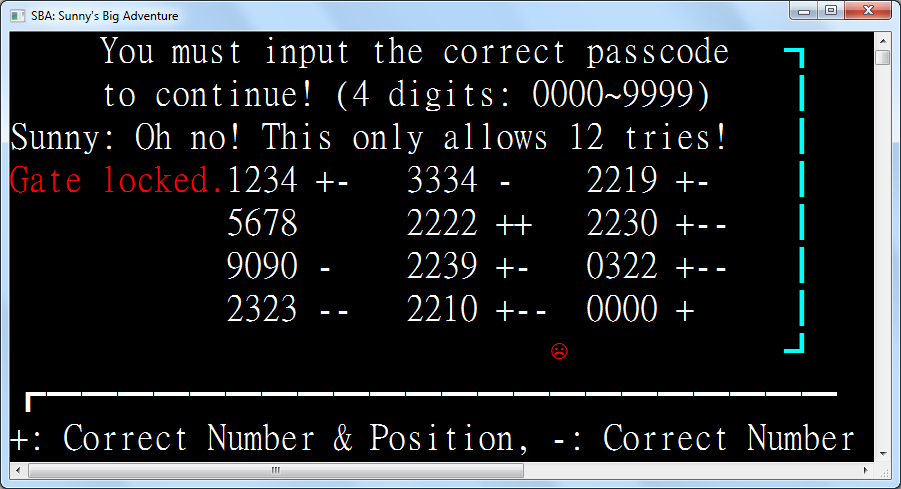
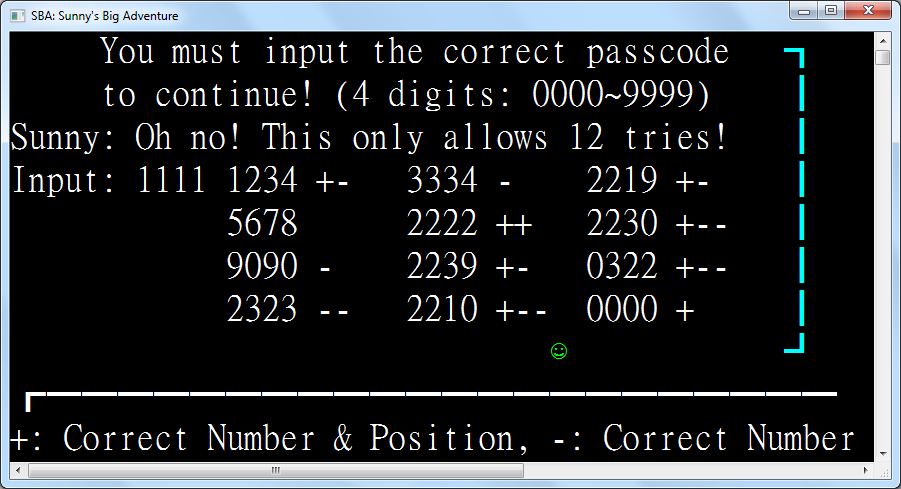
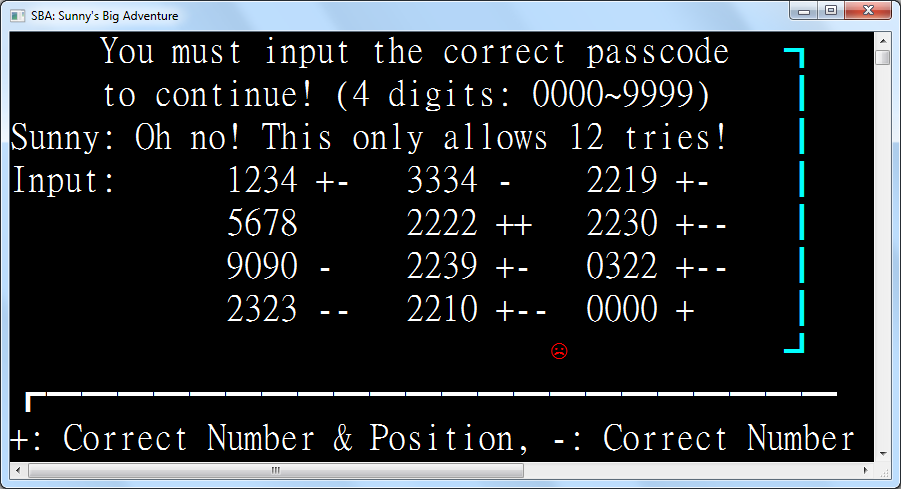
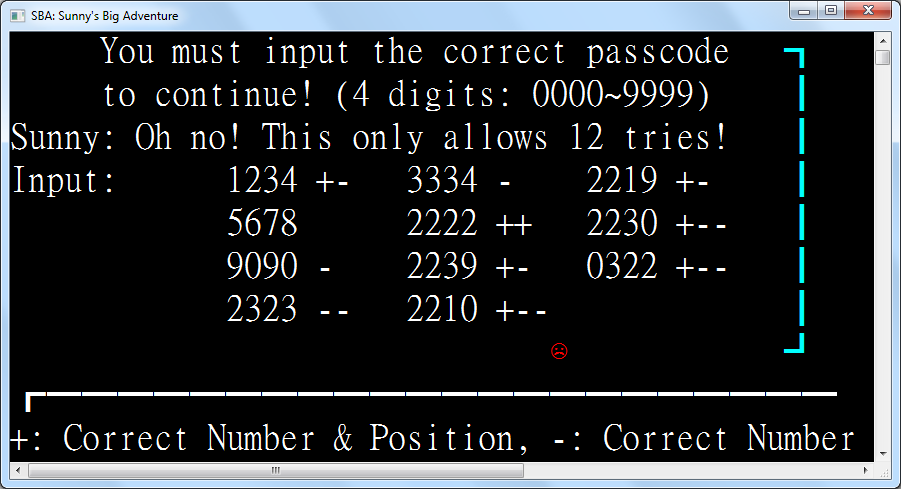
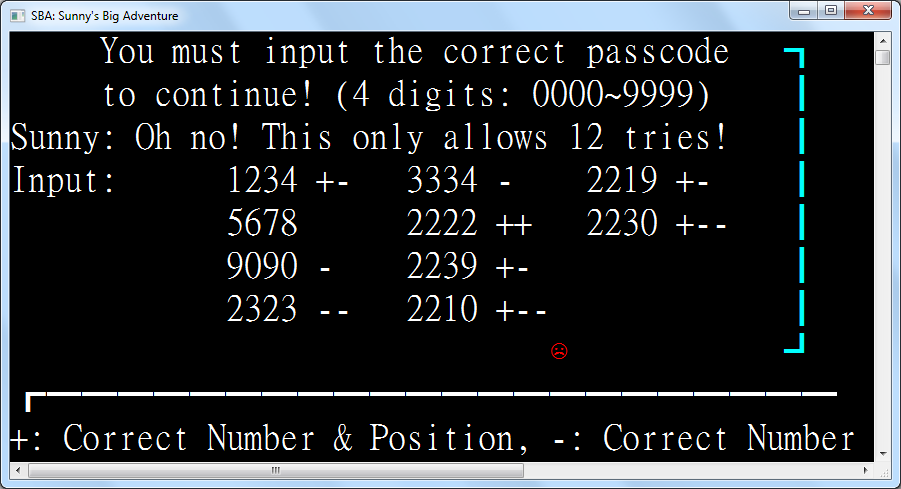
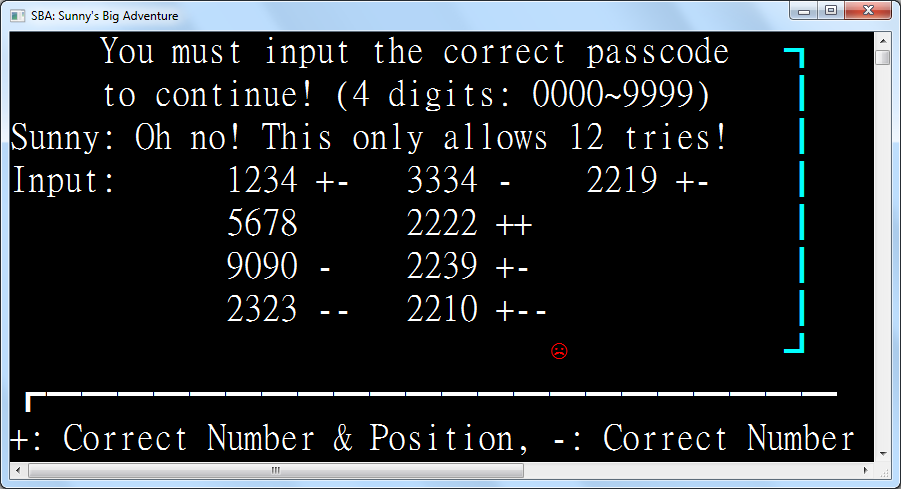
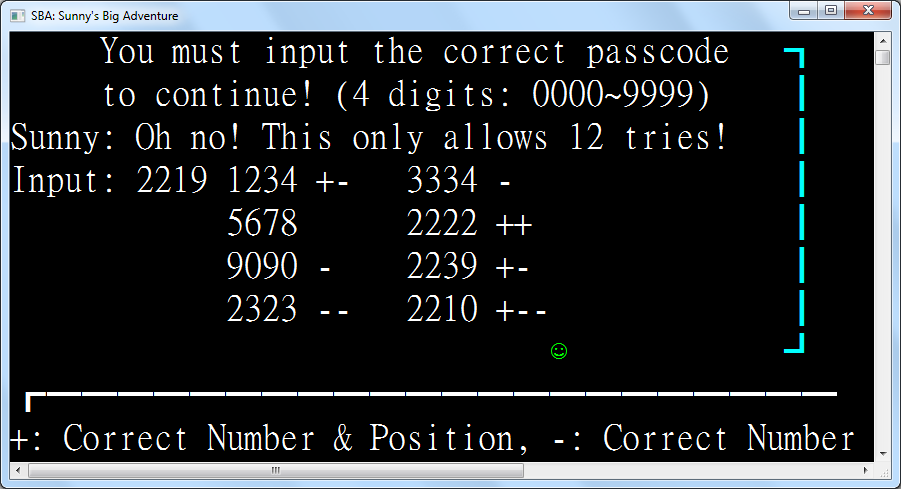
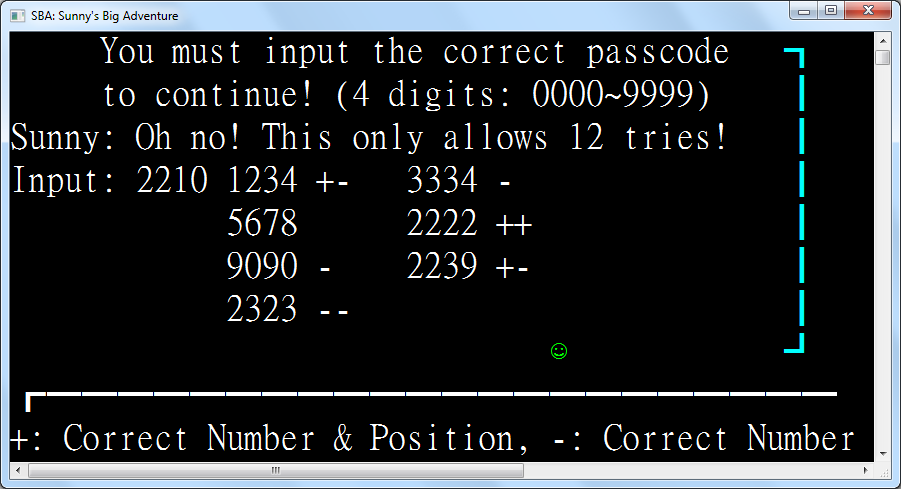
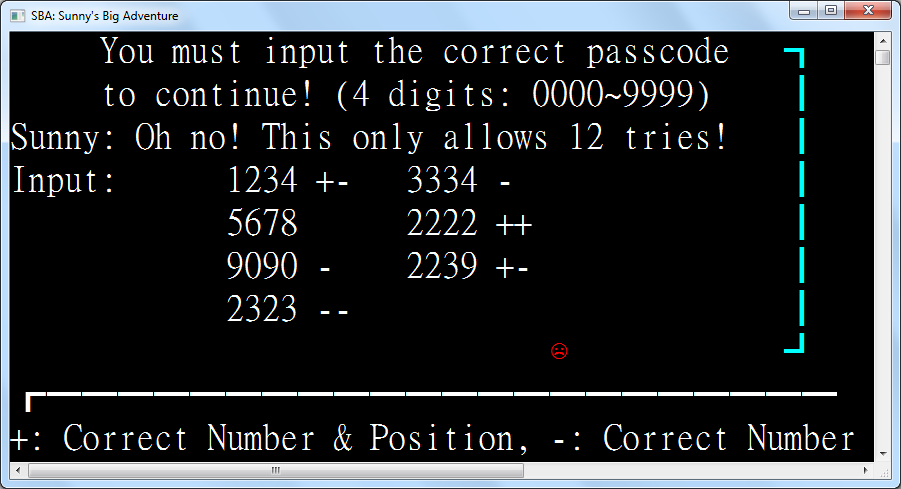
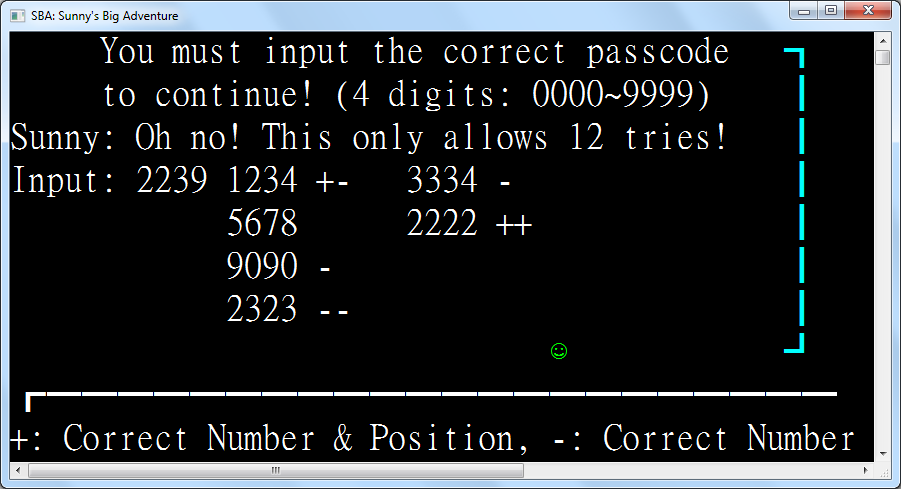
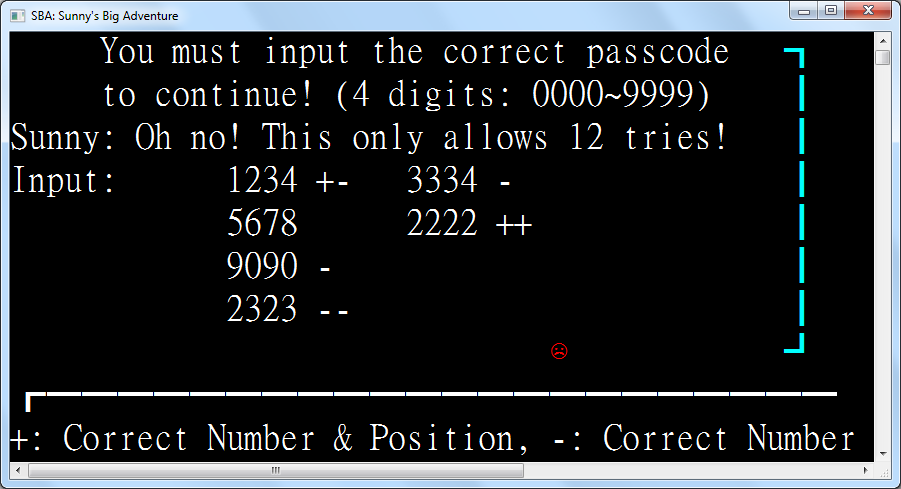
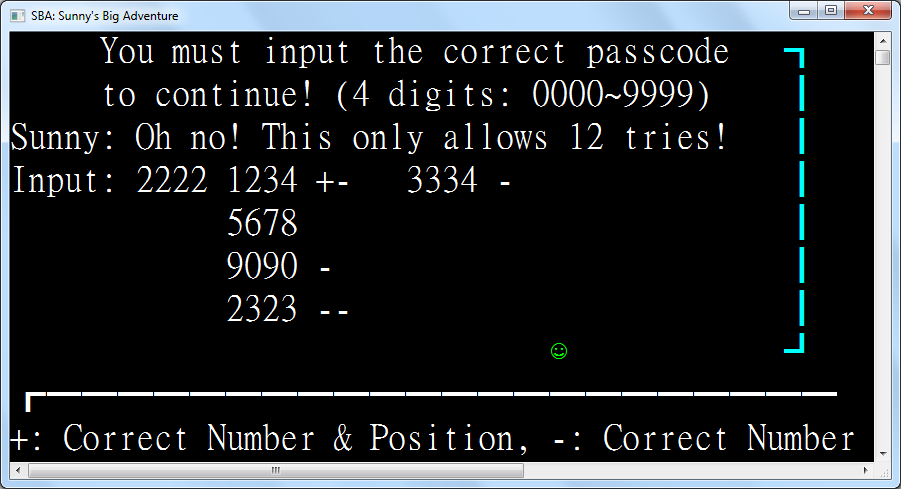
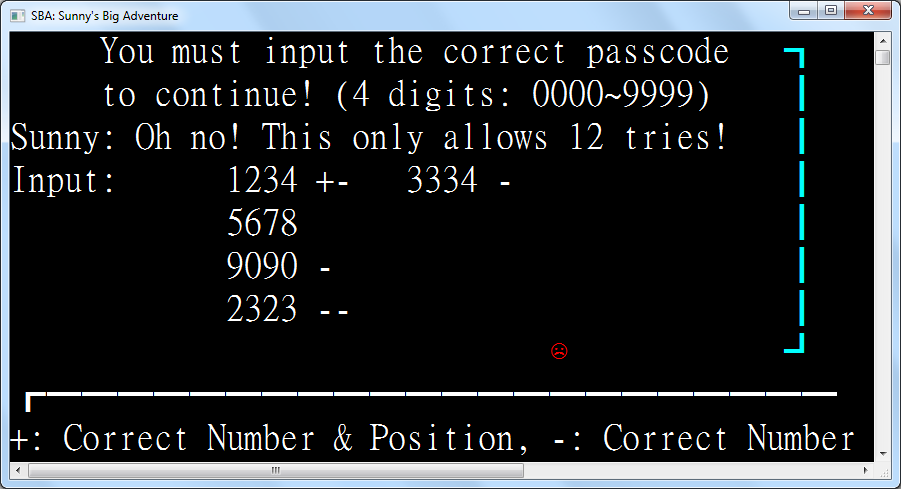
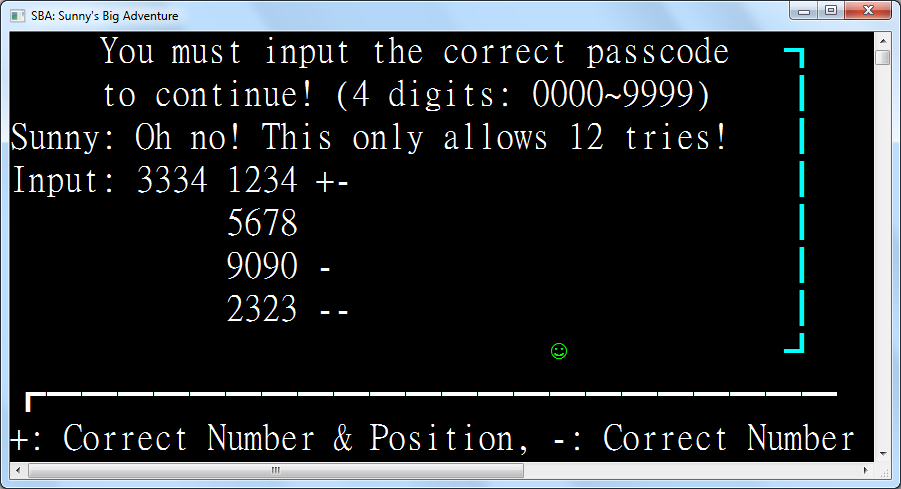
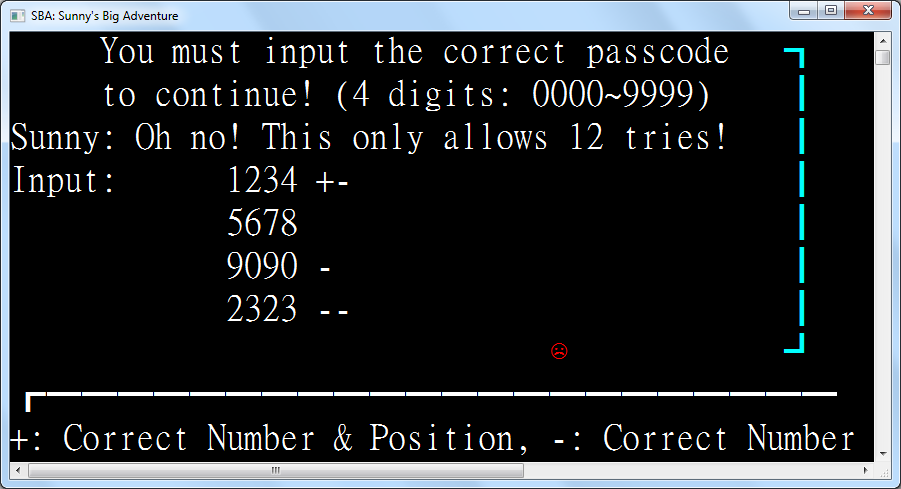
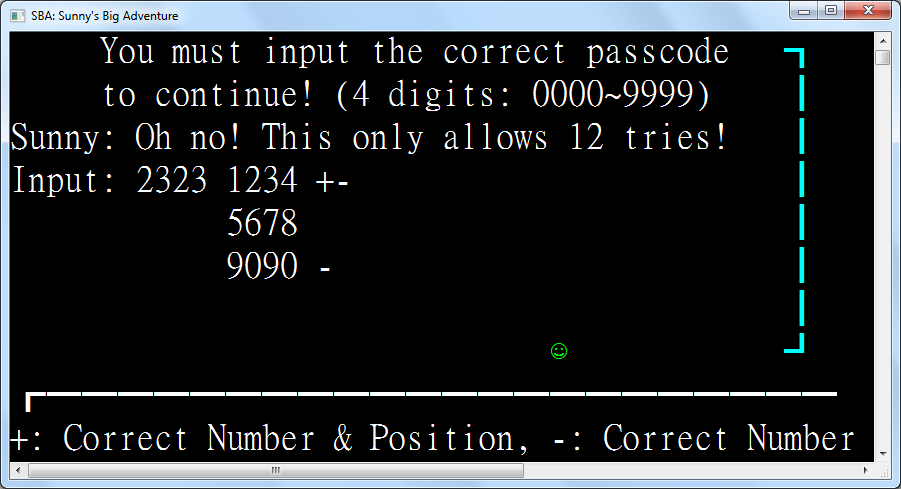
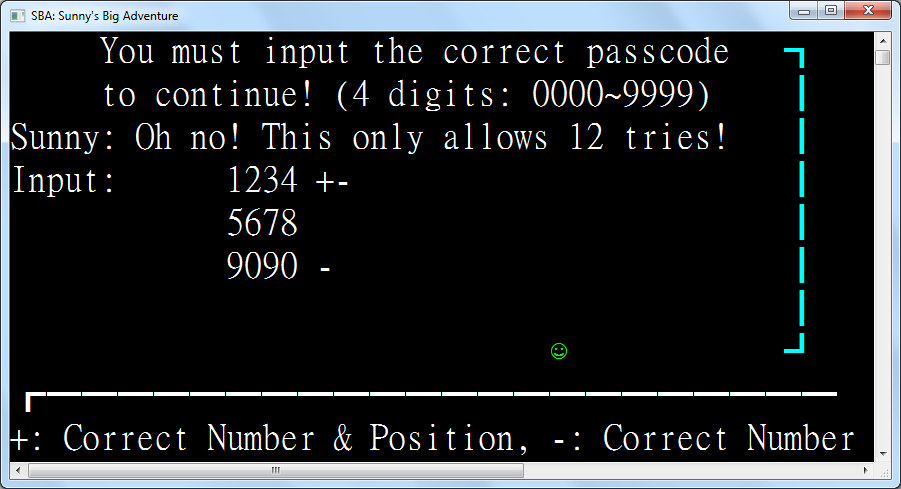
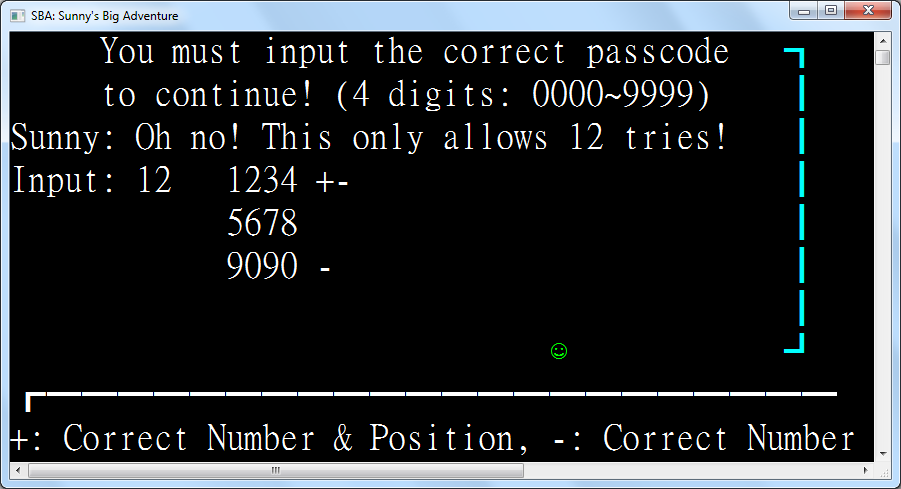
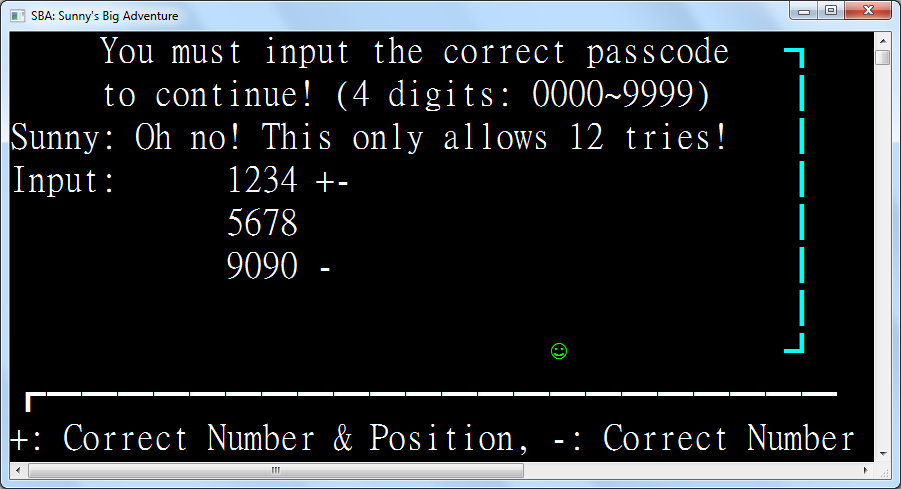
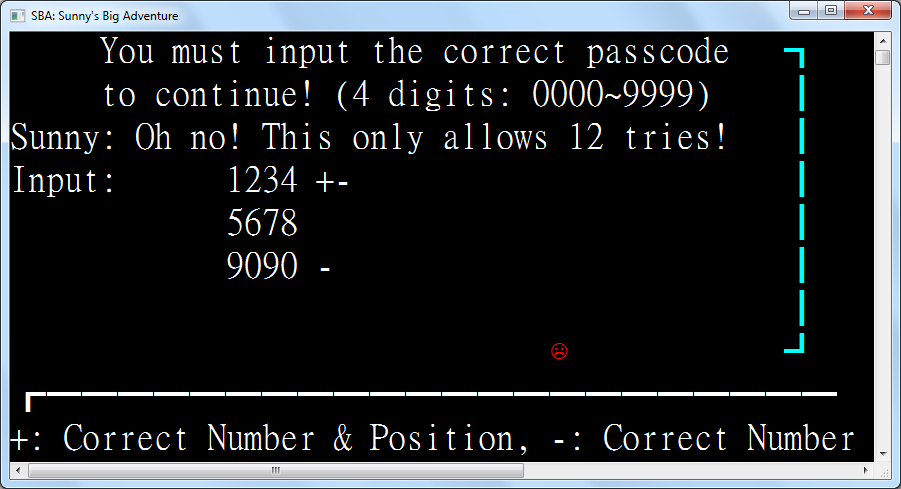
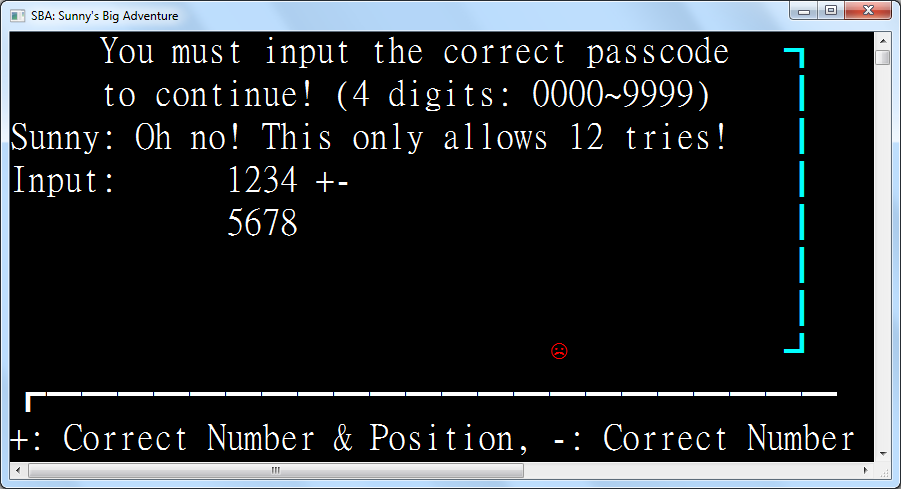
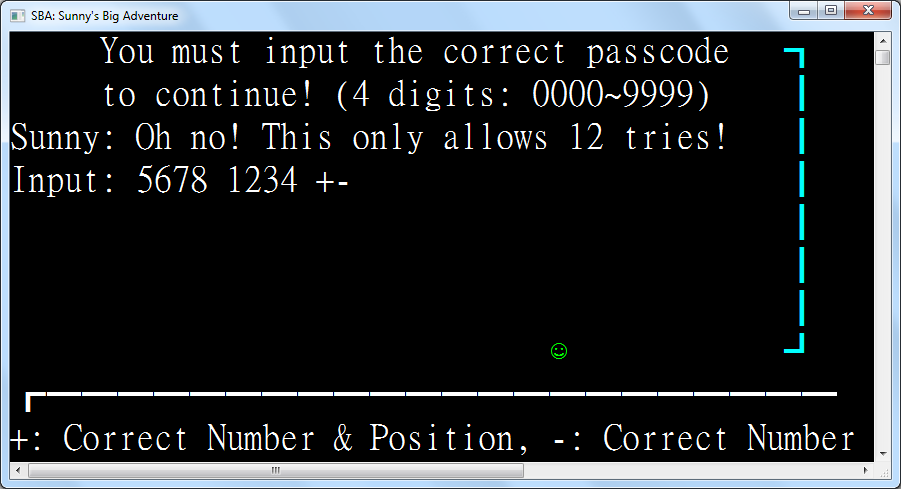
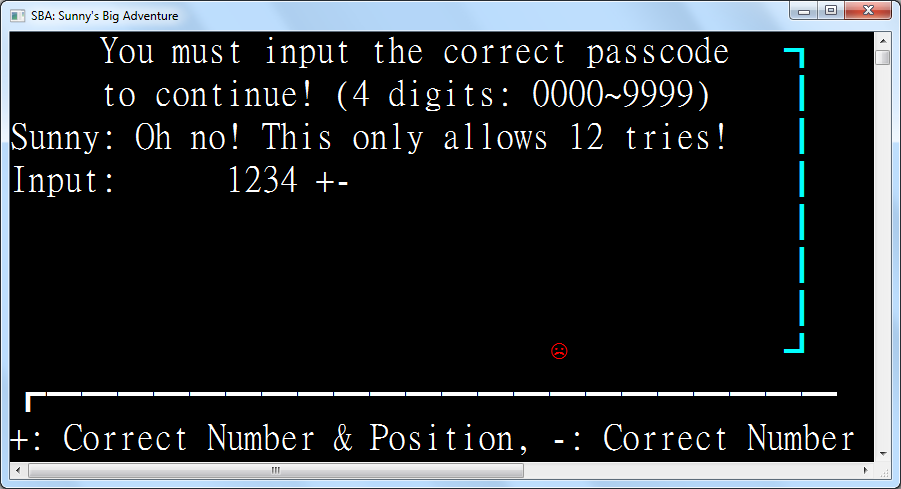
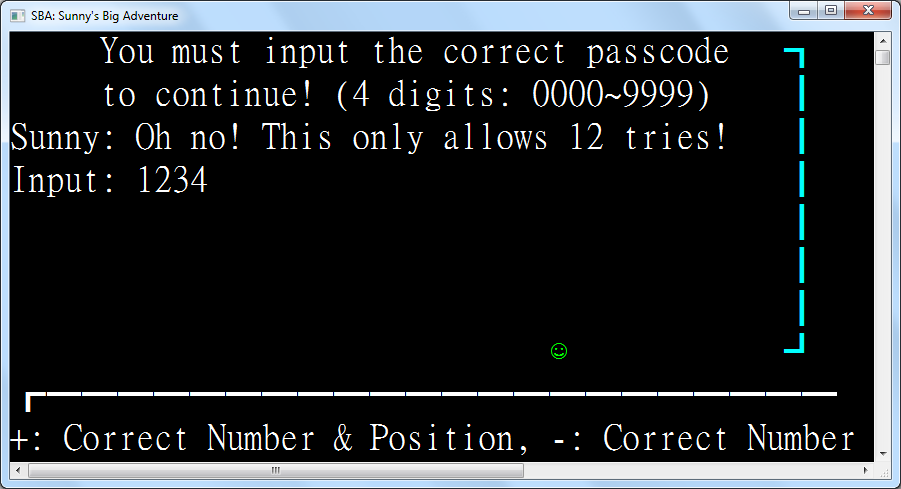
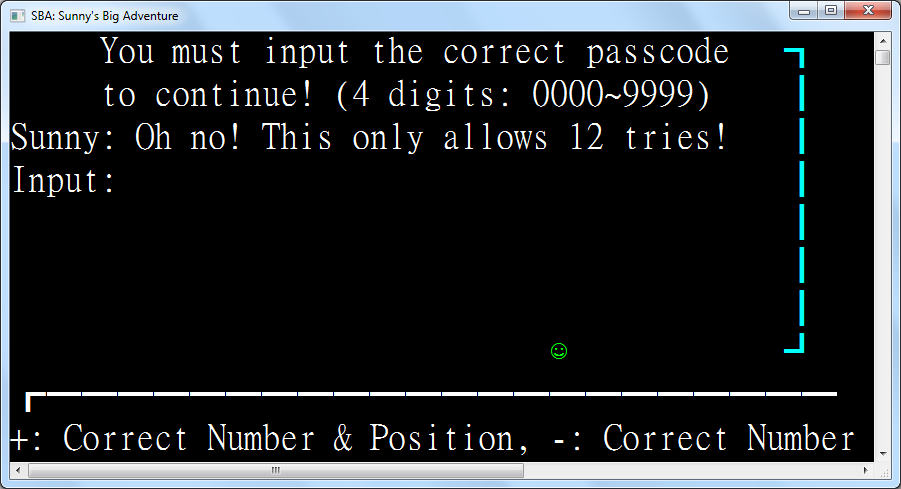
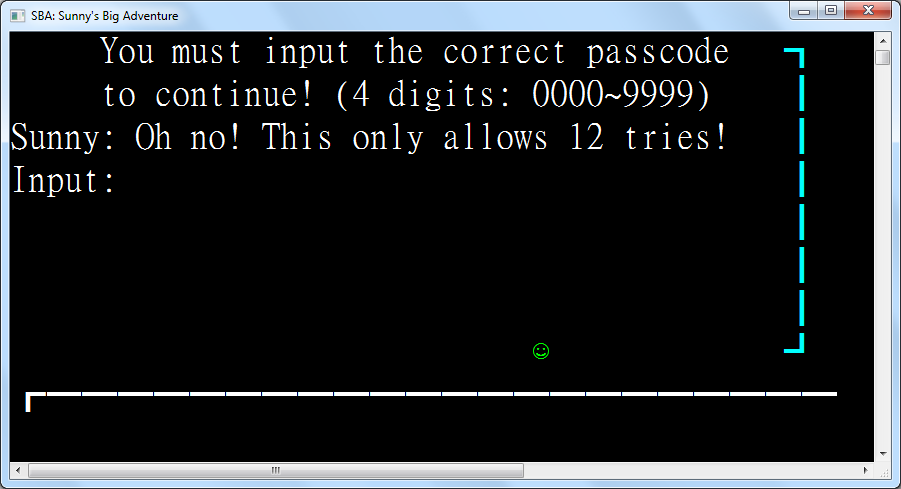
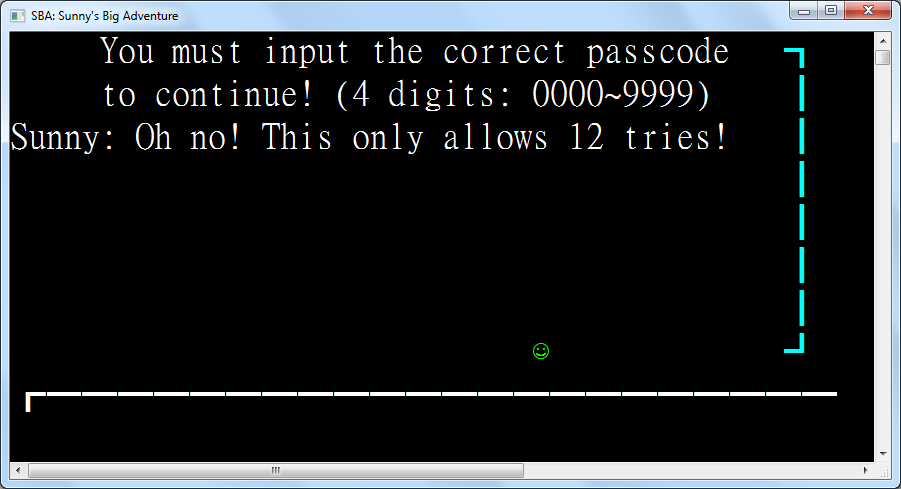
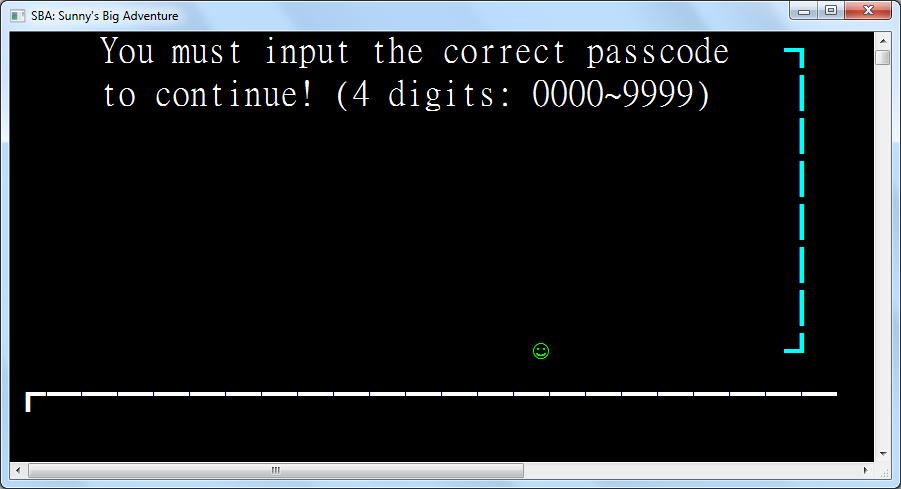
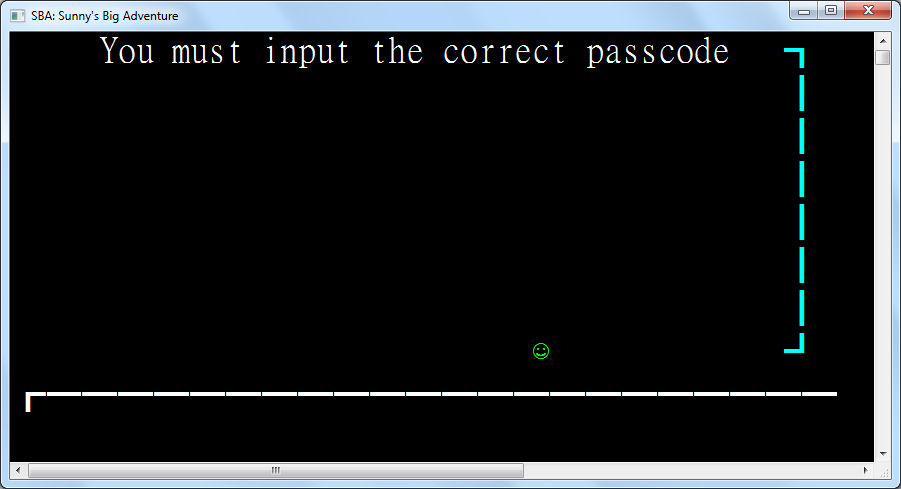
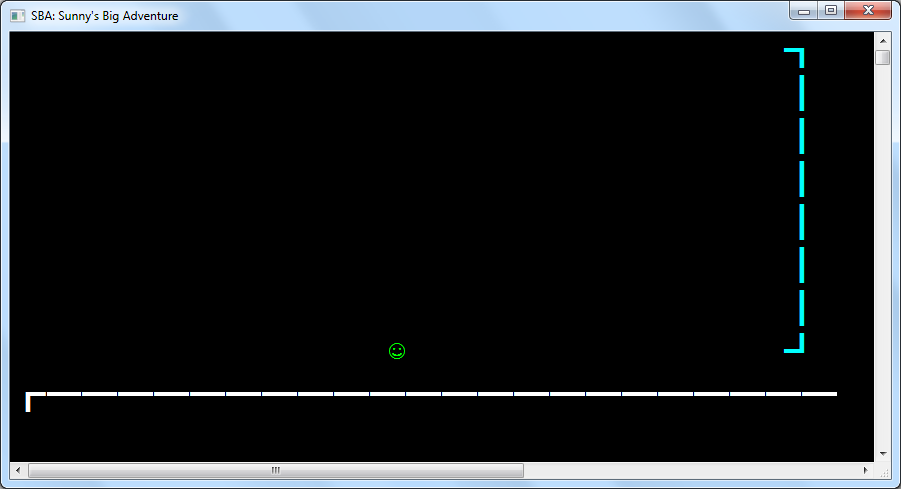
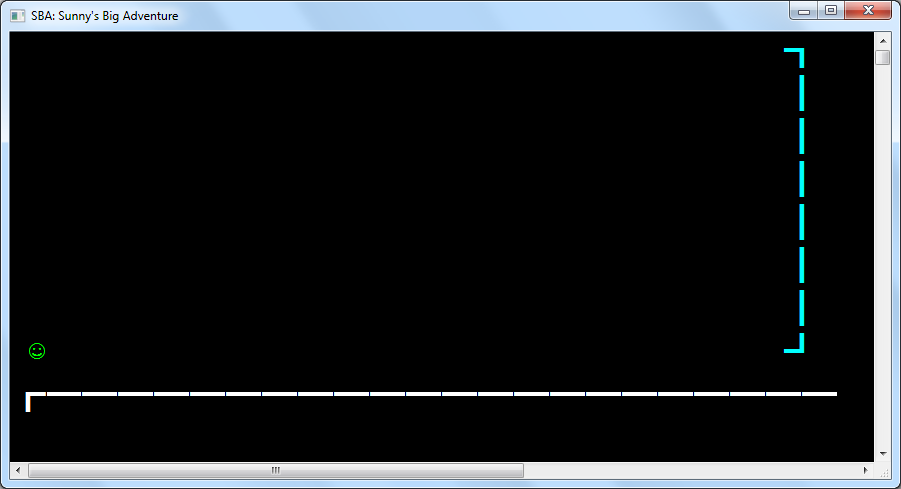
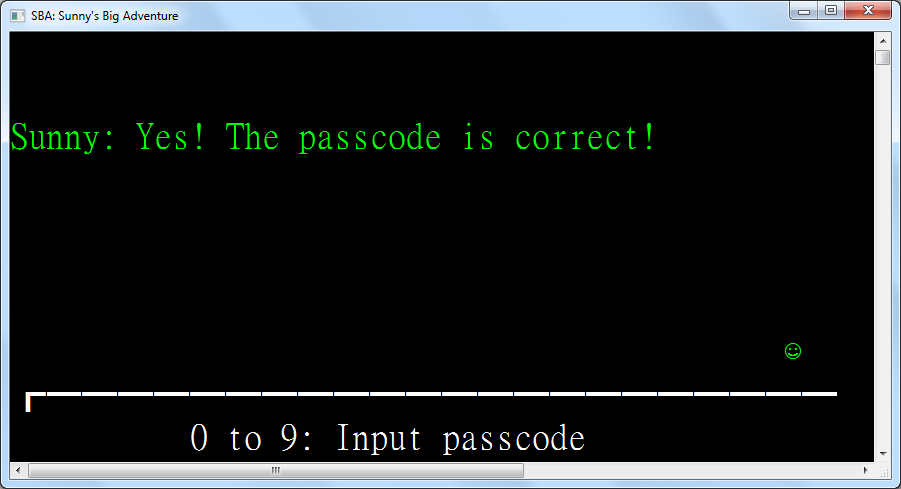
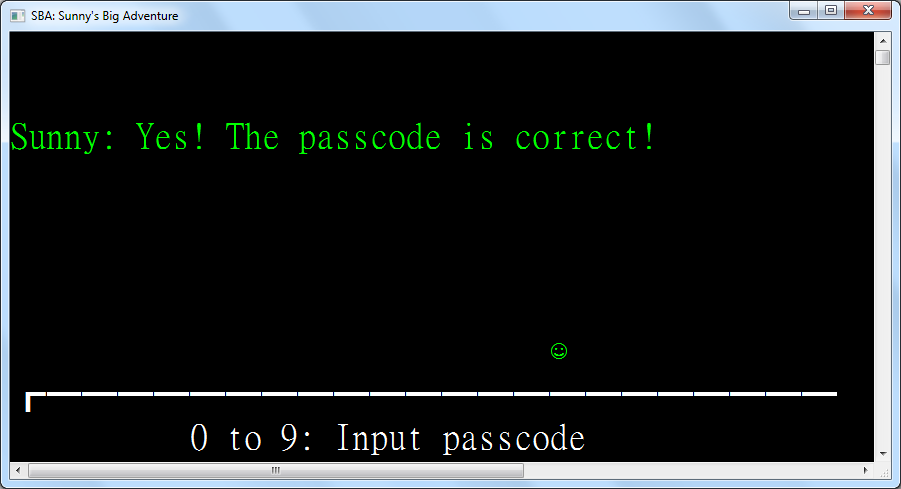
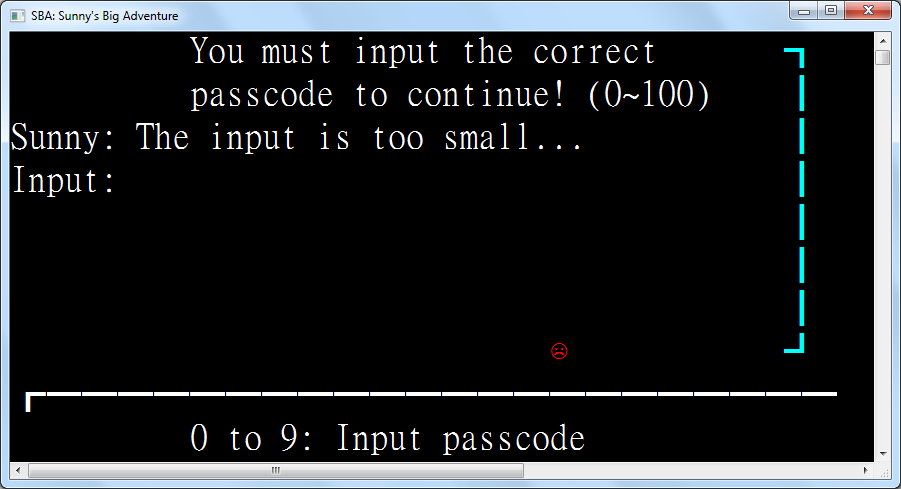
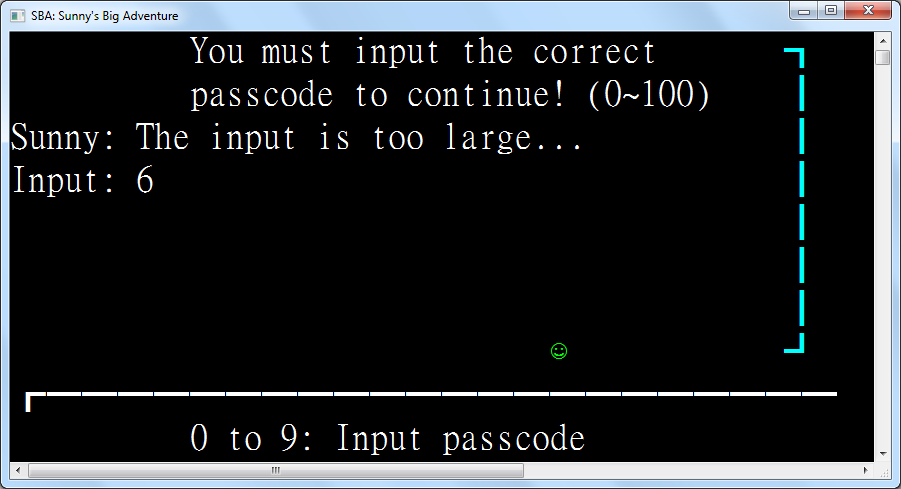
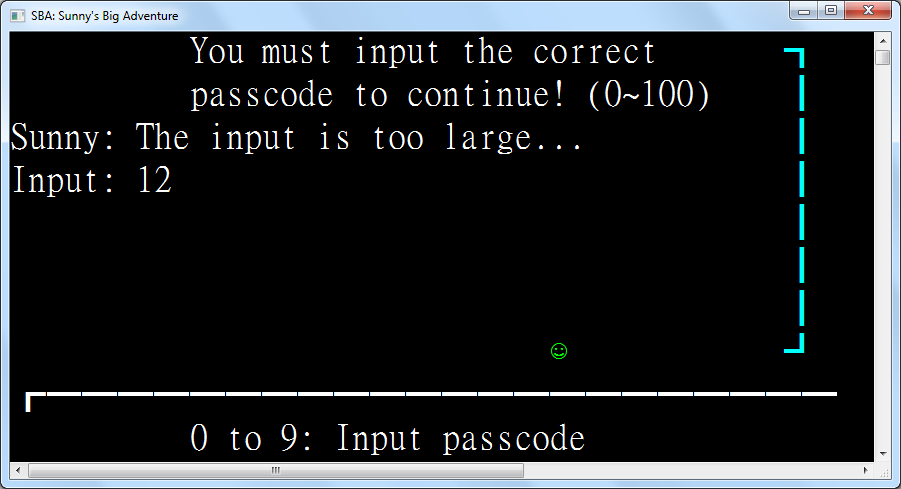
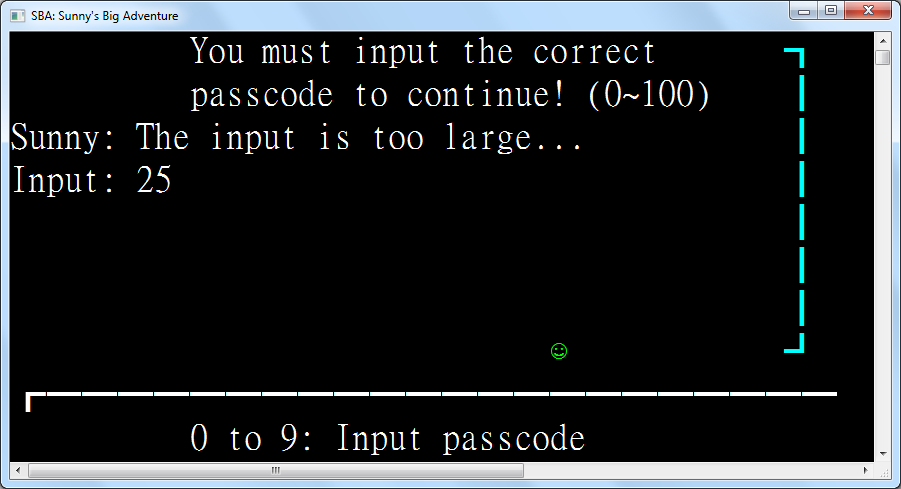
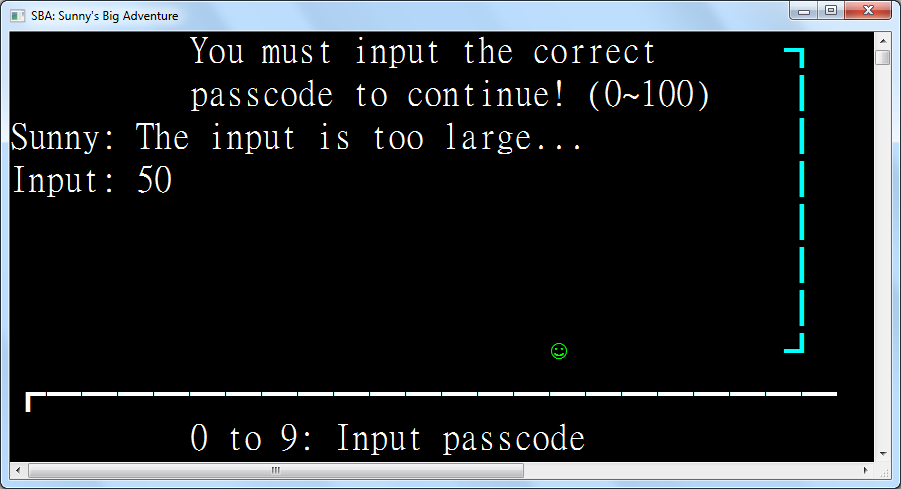
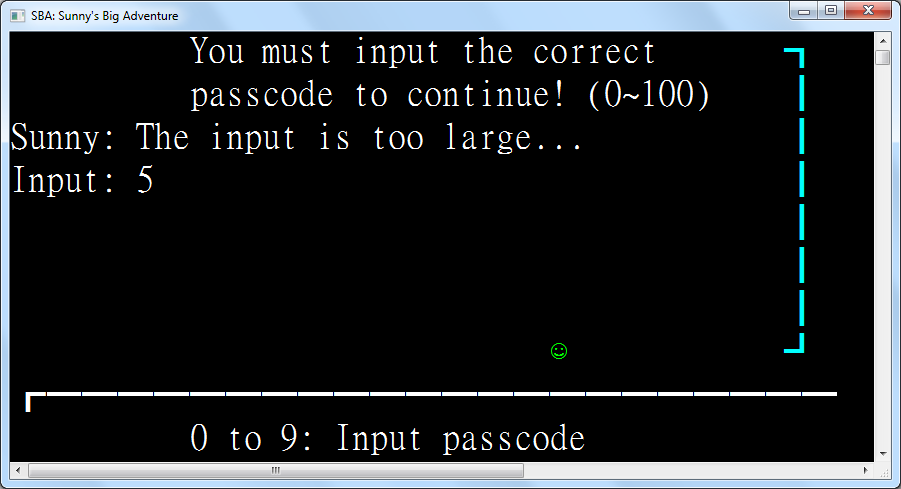
This is the first region in the system. This part will explain the details inside this region.

Process of the Procedure:



1) Input '1' in the menu so as to select the game Peaceful War. Once the





**Chapter 4: System Testing and Evaluation**

**4.1: The User Interface**

In this chapter, testing and evaluation of different aspects in the system will be shown.

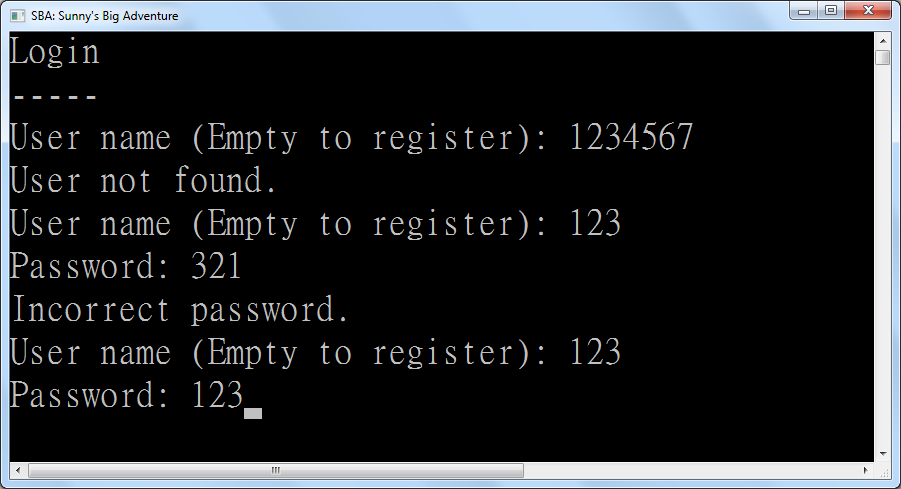
The following will be shown:

1. Test Cases Design and Test Results
2. Further Improvements to the system

**4.2: Test Cases Design and Results**

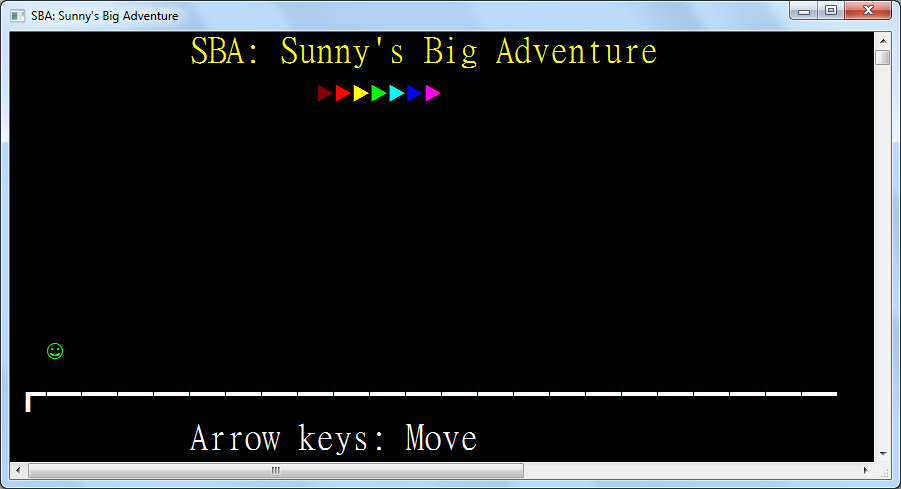
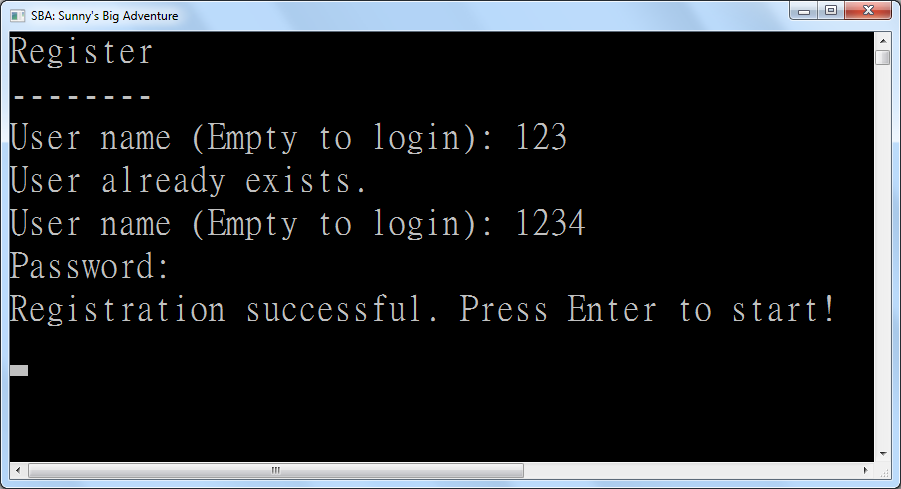
In this sub-chapter, testing is divided for each subprogram where test cases will first be described, then the expected outcome and the test results, and finally whether it matches the expected outcome and problems encountered if any. The details of the testing are as follows.

**4.2.1: Tests for Login**



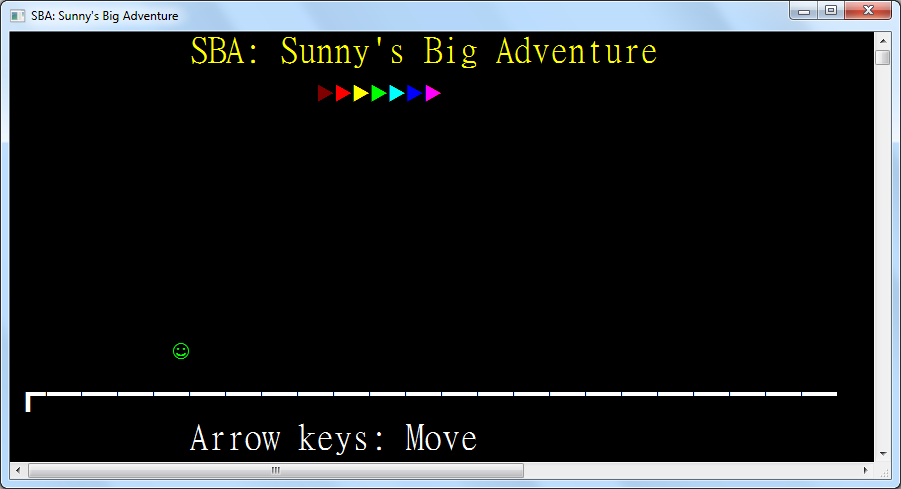
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Name: 1234567 | Invalid Input | Display “User not found” | As expected |
| Name: 123  Pass: 321 | Invalid Input | Display “Incorrect password” | As expected |
| Name: 123  Pass: 123 | Valid Input | Successfully login | As expected |
| Name: 123  Empty Pass | Invalid Input | Display “Incorrect password” | As expected |
| Empty Name | Valid Input | Transfer to Registration Page | As expected |

**4.2.2: Tests for Registration**



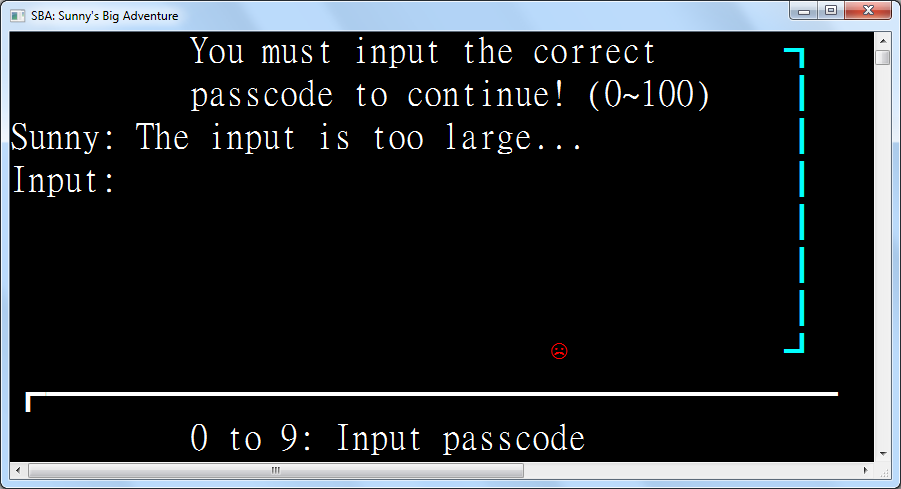
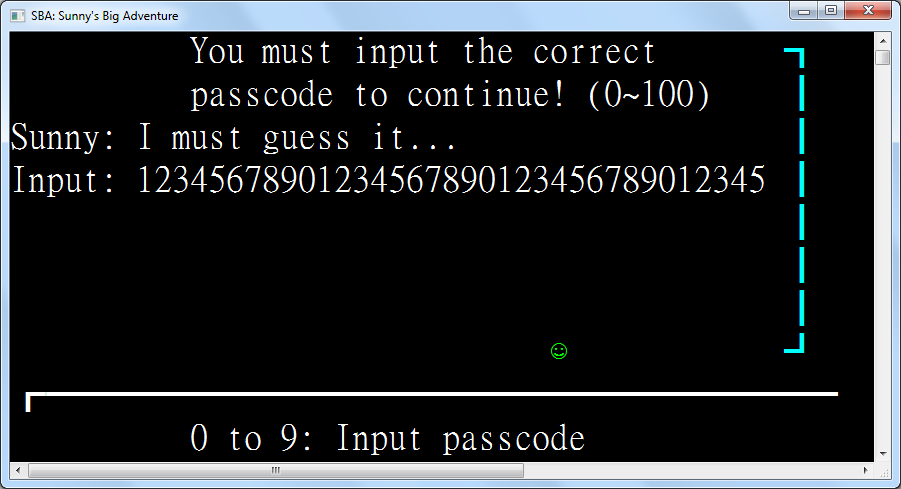
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Name: 123 | Invalid Input | Display “User already exists” | As expected |
| Name: 1234  Empty PW | Valid Input | Successfully register | As expected |
| Name: 12345  Pass: 12345 | Valid Input | Successfully register | As expected |
| Name: ~!@#$%^&\*()\_+  Pass: +\_)(\*&^%$#@!~ | Valid Input | Successfully register | As expected |
| Empty Name | Valid Input | Transfer to Login Page | As expected |

**4.2.3: Tests for Title Screen**



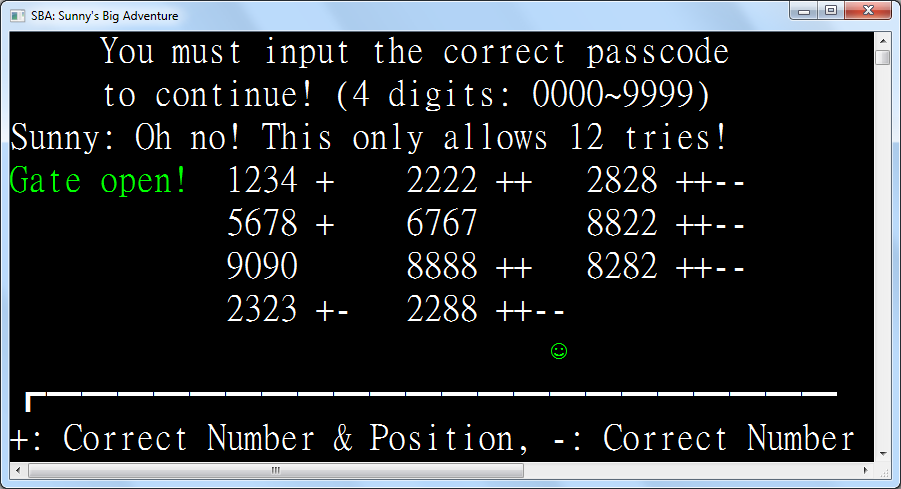
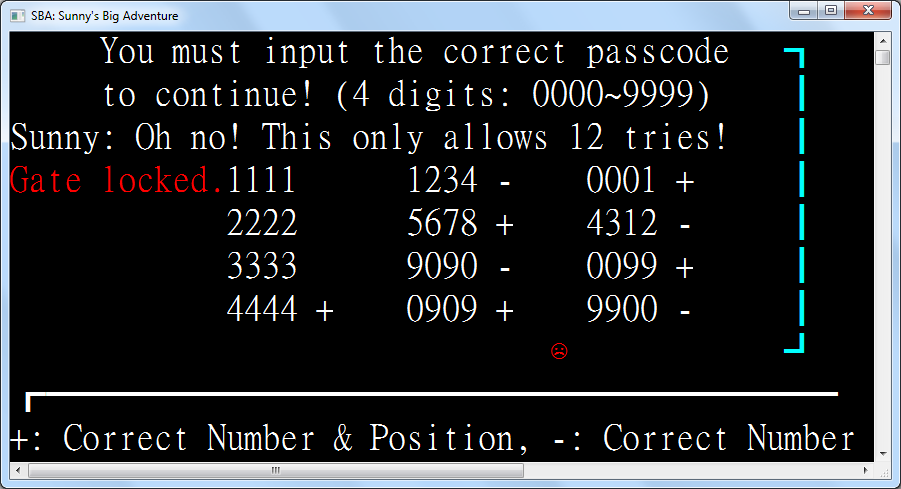
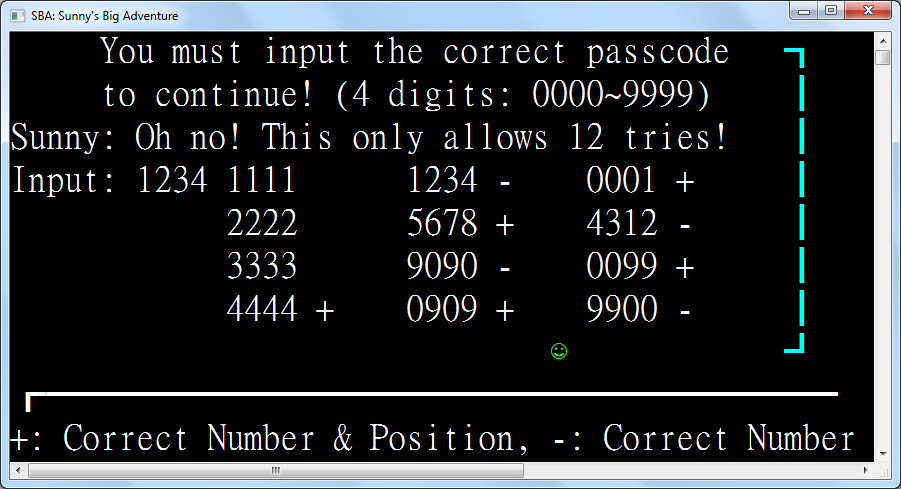
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Left arrow | Valid Input | Avatar moves left | As expected |
| Right arrow | Valid Input | Avatar moves right | As expected |
| Up arrow | Valid Input | Avatar jumps and is pulled down by gravity | As expected |
| Down arrow when avatar is in air | Valid Input | Avatar drops faster | As expected |
| The 1 key | Invalid Input | Nothing happens | As expected |
| Backspace | Invalid Input | Nothing happens | As expected |
| Escape | Invalid Input | Nothing happens | As expected |

**4.2.4: Tests for Number Guess**



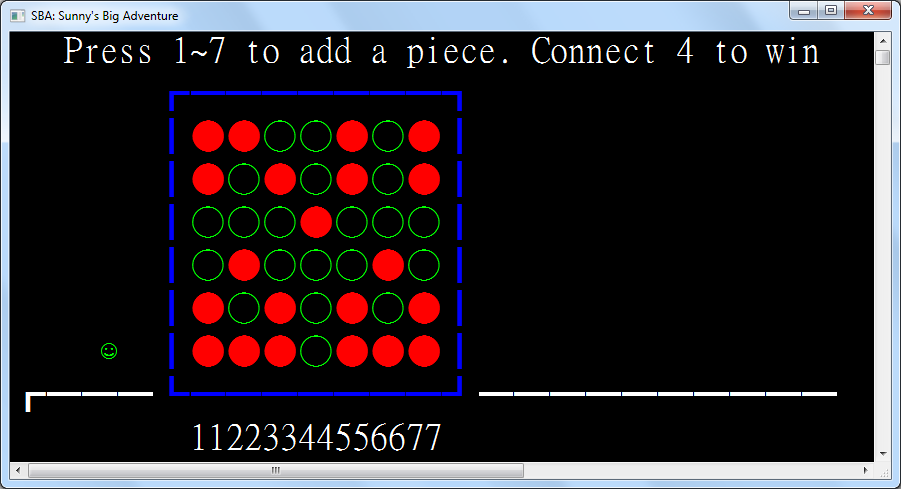
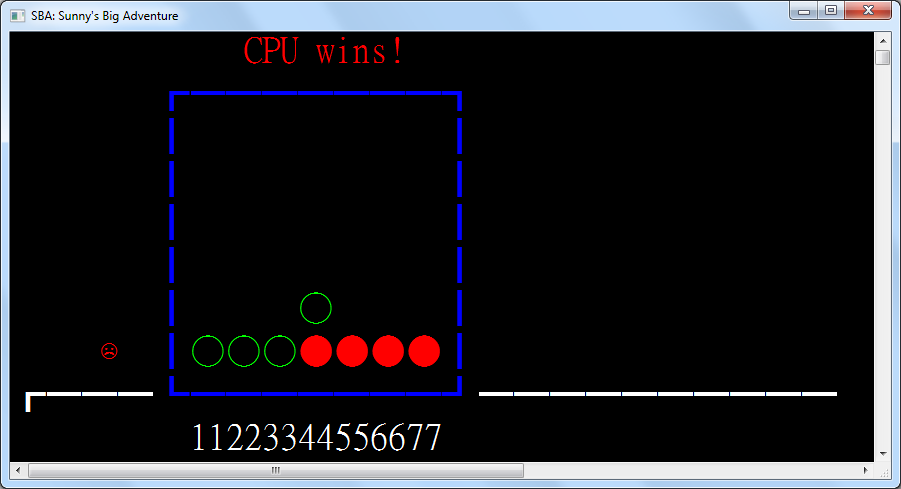
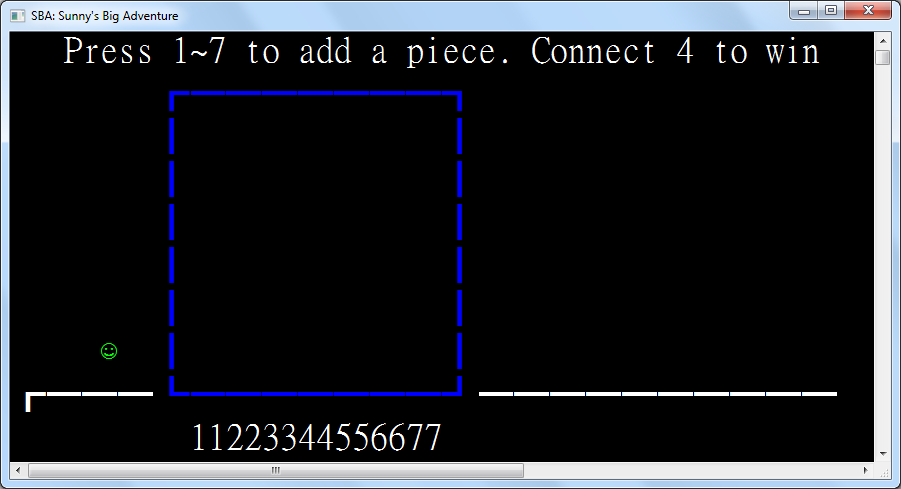
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| 12345678901234567890123456789012345 | Valid Input | Display “Input too large” | As expected |
| 0 | Valid Input | Display “Input too small” | As expected |
| -1 | Invalid Input | The minus sign is ignored | As expected |
| abc | Invalid Input | Keys are ignored | As expected |
| 100 | Valid Input | Display “Input too large” | As expected |
| Empty | Extreme input | Nothing happens | “Input too large” is displayed |
| Backspace when input is present | Valid Input | Deletes the last inputted character if any | As expected |
| Backspace when input is empty | Extreme input | Nothing happens | As expected |
| Escape key | Invalid Input | Nothing happens | As expected |

**4.2.5: Tests for Bulls and Cows**



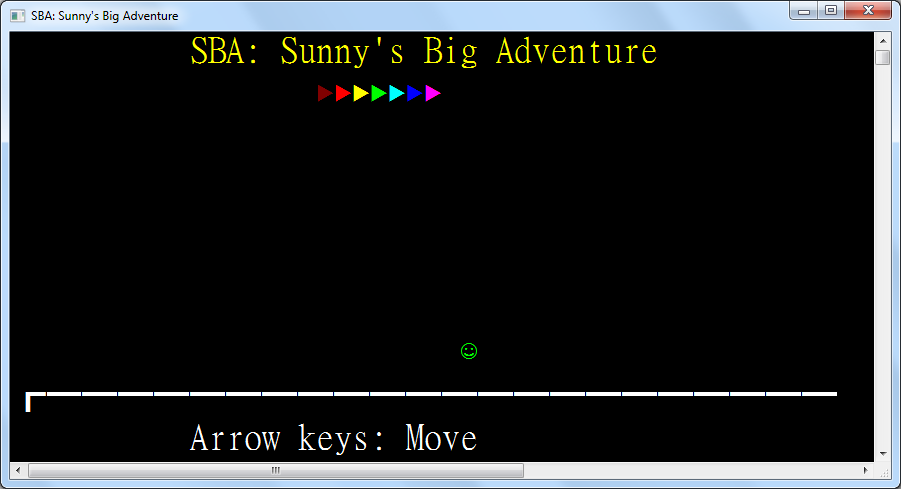
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| 1111 | Extreme Input | Display +s and –s depending on the digits | As expected |
| 1234 | Valid Input | Display +s and –s depending on the digits | As expected |
| 5678 | Valid Input | Display +s and –s depending on the digits | As expected |
| 9090 | Valid Input | Display +s and –s depending on the digits | As expected |
| 0000 | Extreme Input | Display +s and –s depending on the digits | As expected |
| Empty | Invalid Input | Input is ignored and reset | As expected |
| 1 | Invalid Input | Input is ignored and reset | As expected |
| 123 | Invalid Input | Input is ignored and reset | As expected |
| -1234 | Invalid Input | Minus sign is ignored | As expected |
| 12345678 | Invalid Input | Digits after the 4th one are ignored | As expected |
| Backspace when input is present | Valid Input | Deletes the last inputted character if any | As expected |
| Backspace when input is empty | Extreme input | Nothing happens | As expected |
| Escape key | Invalid Input | Nothing happens | As expected |

**4.2.6: Tests for Connect Four**



|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Key 1 | Extreme Input | Player puts a piece at position 1 | As expected |
| Key 4 | Valid Input | Player puts a piece at position 4 | As expected |
| Key 7 | Extreme Input | Player puts a piece at position 7 | As expected |
| Key 8 | Invalid Input | Nothing happens | As expected |
| Key 0 | Invalid Input | Nothing happens | As expected |
| Backspace key | Invalid Input | Nothing happens | As expected |
| Escape key | Invalid Input | Nothing happens | As expected |
| Player places 4 pieces horizontally | Valid Input | Player wins | As expected |
| CPU places 4 pieces horizontally | Valid Input | CPU wins | As expected |
| Player places 4 pieces vertically | Valid Input | Player wins | As expected |
| CPU places 4 pieces vertically | Valid Input | CPU wins | As expected |
| Player places 4 pieces diagonally | Valid Input | Player wins | As expected |
| CPU places 4 pieces diagonally | Valid Input | CPU wins | As expected |
| Player places 3 pieces horizontally | Valid Input | CPU places piece to stop the player from winning | As expected |
| Player places 3 pieces vertically | Valid Input | CPU places piece to stop the player from winning | As expected |
| Player places 3 pieces diagonally | Valid Input | CPU places piece to stop the player from winning | As expected |
| Player places piece with no immediate chance of winning | Valid Input | CPU places piece randomly | As expected |

**4.2.7: Tests for Winning Screen**



|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Left arrow | Valid Input | Avatar moves left | As expected |
| Right arrow | Valid Input | Avatar moves right | As expected |
| Up arrow | Valid Input | Avatar jumps and is pulled down by gravity | As expected |
| Down arrow when avatar is in air | Valid Input | Avatar drops faster | As expected |
| Enter | Valid Input | Player is taken back to the Title Screen | As expected |
| The 1 key | Invalid Input | Nothing happens | As expected |
| Backspace | Invalid Input | Nothing happens | As expected |
| Escape | Invalid Input | Nothing happens | As expected |
| Player jumps to a firework | Valid input | Player is blocked from overlapping with the firework | As expected |

<https://docs.google.com/forms/d/e/1FAIpQLSdV4dhhAfih9IGHCtSLh1LIMF4F4i77OjfYHD4yy1ADs-iEUw/viewform>

**Chapter 5:**