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:

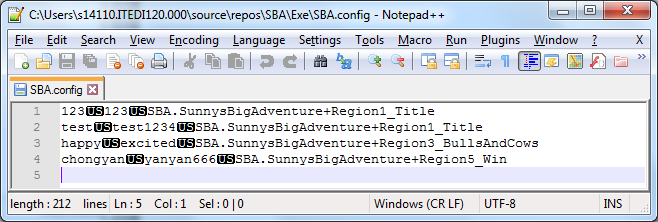
1. Database Implementation

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.

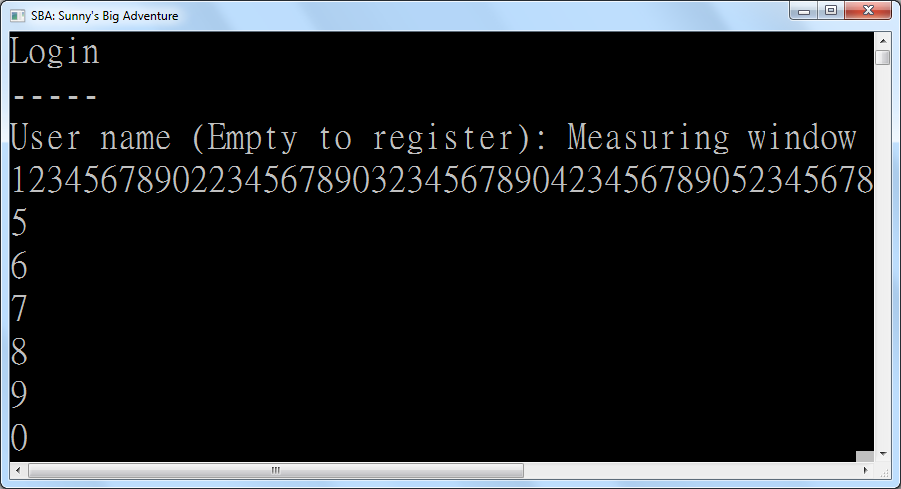
When the array is searched linearly, each string will be split by the Unit Separator and the relevant field will be extracted and used.

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.

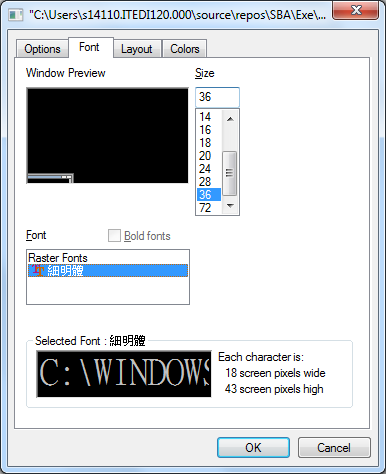
The array will never be saved in the program memory to ensure that there is only one source of truth: the file, to avoid bugs arising from inconsistencies between the file and the program memory. Other than that, this approach can also ensure a more pleasant debugging experience where changes in the 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.

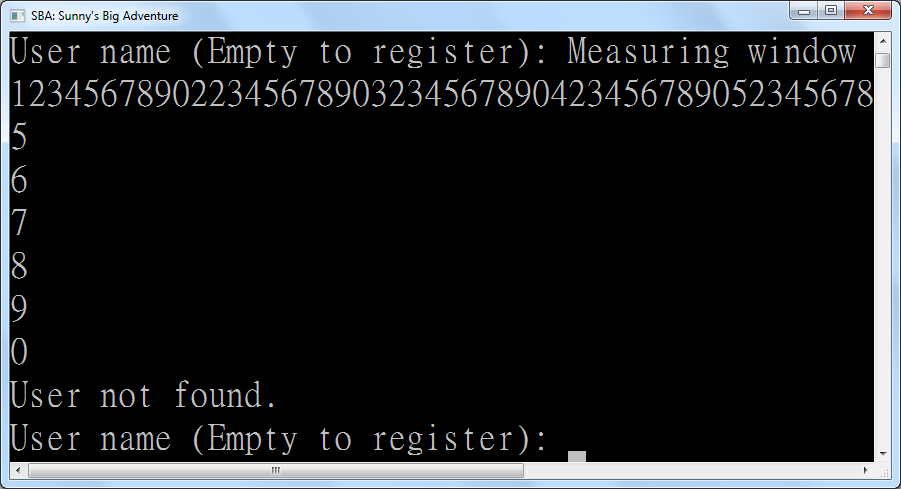


There is a requirement for the font used. It should be 細明體with a large font size because it provides large and beautiful characters and emojis, especially the happy face and the sad face for the avatar.

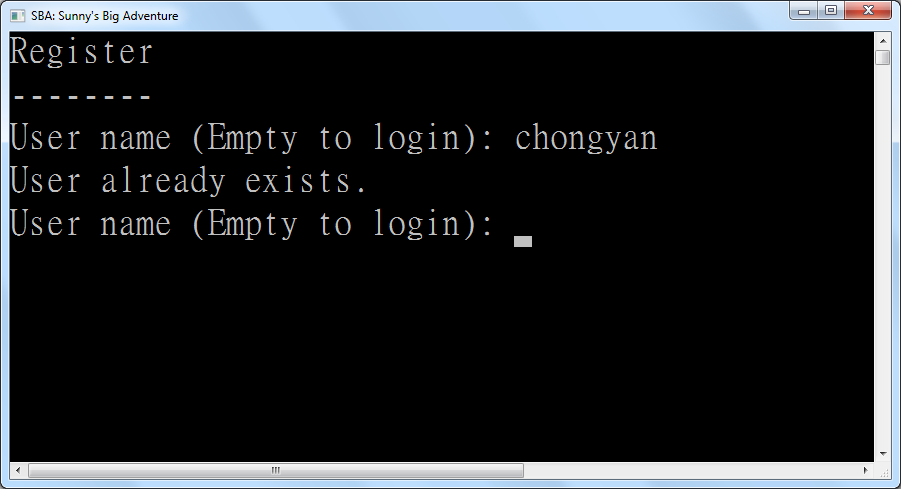


The console window is used to give instructions and guidelines to its users. Users can simply follow the instructions and guidelines given by the system and input movement keys or input keys in order to enjoy the game.

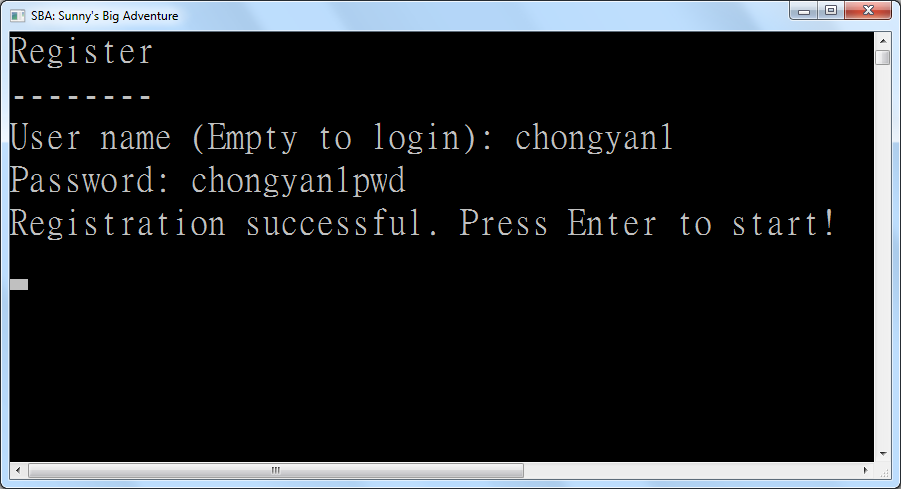
If errors related to invalid input occur, namely “User not found” and “Incorrect password” for login, as well as “User already exists” for registration, that error will be displayed to the user for inputting the data again. This will continue forever until the user finally provides valid data.



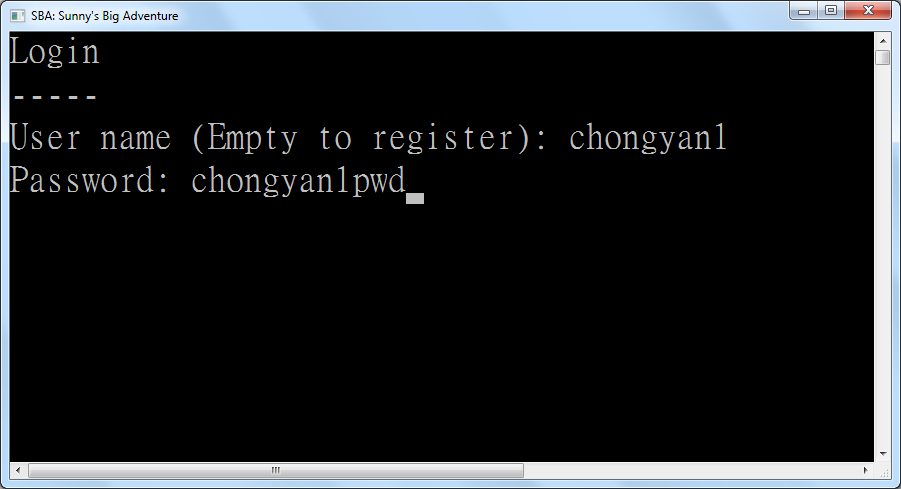
To eliminate the use of user-unfriendly “y” or “n” inputs, empty input is treated as the command for switching between login and registration screens. This way, less time is spent on inputting decisions that the user rarely inputs, like whether to register a new account, as each person usually only ever 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 with source code in the Visual Basic .NET programming language. 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 property provides the area this entity is currently occupying. It can be null which indicates that this entity is currently unloaded and off-screen. If assigned a new value, this property will automatically check for whether the new position is available to be moved to and redraw the entity at the new location in order to simplify moving the entity, reuse code and prevent bugs where the stored location and the drawn position are inconsistent.

|  |
| --- |
| Public Property Position As Point?  Get  Return Bounds?.TopLeft  End Get  Set(value As Point?)  Bounds = IfHasValue(value, AddressOf BoundsForNewPoint)  End Set  End Property |

The Position property is a convenient way to update the top-left point of the Bounds property so that whenever an entity moves, a whole new Rectangle need not be constructed in the calling code, which is prone to bugs where the size of the entity is inconsistent.

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

The BoundsForNewPoint function is marked as MustOverride to force each subclass to implement different measuring code which is unique to each subclass.

|  |
| --- |
| ''' <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 |

The Go functions are convenient ways to move the entity by one unit so that excessive use of point constructors will not be needed, enabling conciseness of code.

|  |
| --- |
| Public Overridable Sub Dispose() Implements IDisposable.Dispose  Position = Nothing  End Sub |

The Dispose function subroutine defaults to setting the position to null, unloading this entity and making it disappear. However, subclasses can override this subroutine to implement their own way of unloading, as seen in trigger zones.

|  |
| --- |
| End Class |

**3.6.2: Rectangle Entity**

|  |
| --- |
| 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 |

A RectangleEntity is a rectangle on screen, like the ground. This constructor accepts a list of entities, the rectangular bounds, and optionally the rectangle color, defaulting to white.

|  |
| --- |
| 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 |

The ForbidEntry override checks whether the incoming entity overlaps with any of this entity’s sides. This means that RectangleEntitys are hollow. The reason why they are not solid despite being easier to implement is that in the Connect Four game, pieces are dropped inside a blue rectangle entity, which acts as the game’s outline.

|  |
| --- |
| Protected Overrides Function BoundsForNewPoint(point As Point) As Rectangle?  Return IfHasValue(Bounds, Function(rect) New Rectangle(point, rect.Width, rect.Height))  End Function |

The BoundsForNewPoint override supplies the new rectangular area for a possible new location.

|  |
| --- |
| 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 |

The BoundsForNewPoint override supplies the new rectangular area for a possible new location.

|  |
| --- |
| 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 |

The RedrawAt override draws the colored rectangle on-screen with matching Unicode box-drawing characters, after erasing the old drawn rectangle on the screen by overwriting it with spaces.

|  |
| --- |
| End Class |

**3.6.3: Trigger Zone**

|  |
| --- |
| Class TriggerZone  Inherits RectangleEntity  Public Sub New(entities As ICollection(Of Entity), rect As Rectangle?,  Optional keyPress As Func(Of ConsoleKey, Boolean) = Nothing,  Optional enter As Action = Nothing, Optional leave As Action = Nothing)  MyBase.New(entities, rect)  Me.Enter = enter  Me.Leave = leave  Me.KeyPress = keyPress  End Sub |

A TriggerZone is an area where an event is raised whenever a player entity enters, leaves, or presses a key while inside the zone. It is the primary way of interaction in games, detecting player input through the keyboard as the medium. This constructor accepts a list of entities, the rectangular bounds, optionally the key press handler, optionally the entry handler, and optionally the exit handler.

|  |
| --- |
| Public Property Enter 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) |

The EnterLock and LeaveLock flags are set when Enter and Leave are raised respectively. It is to prevent these two events from recursively calling themselves when an entity changes state during handling of these events, leading to an infinite loop and the program crashing.

|  |
| --- |
| 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  Enter?.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 |

The ForbidEntry override raises the Enter and Leave events depending on whether the player is entering the trigger zone and exiting the trigger zone respectively.

|  |
| --- |
| Public Overrides Sub Dispose()  If ActiveEntity.Trigger Is Me Then  ActiveEntity.Trigger = Nothing  Leave?.Invoke()  End If  MyBase.Dispose()  End Sub |

The Dispose override raises the Leave event if the player entity is within this trigger zone while it is being unloaded.

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

The RedrawAt override does nothing as the trigger zone is invisible to the player.

|  |
| --- |
| End Class |

**3.6.4: Text Entity**

|  |
| --- |
| 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 |

A TextEntity is a single line of text on screen. This constructor accepts a list of entities, the text, optionally the position of text, and optionally the color of the text.

|  |
| --- |
| Protected Overrides Function BoundsForNewPoint(point As Point) As Rectangle?  Return New Rectangle(point, Text.Length, 1)  End Function |

The BoundsForNewPoint override measures the new bounds by assigning the text length as its width.

|  |
| --- |
| 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 |

The Text property provides a way to change the text of the TextEntity while also updating the displayed text on-screen. This mechanism is used in, for example, number input in Number Guessing.

|  |
| --- |
| 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 |

The RedrawAt override erases the old text by overwriting it with spaces, then writing the new text in the new position on screen.

|  |
| --- |
| End Class |

**3.6.5: Sprite Entity**

|  |
| --- |
| Class SpriteEntity  Inherits Entity  Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)  MyBase.New(entities)  \_sprite = sprite  End Sub |

A SpriteEntity is an entity with a sprite as appearance. This is the base class for TickEntity. This constructor accepts a collection of entities and a sprite as the initial appearance.

|  |
| --- |
| Protected Overrides Function BoundsForNewPoint(point As Point) As Rectangle?  Return New Rectangle(point, 1, 1)  End Function |

The BoundsForNewPoint override measures the new bounds by assigning the width and height 1. This is because each sprite is only one character wide and high.

|  |
| --- |
| 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 |

The Sprite property provides a way to change the sprite of the SpriteEntity while also updating the displayed text on-screen. This mechanism is used in, for example, making the avatar appear sad and angry whenever the player guesses wrongly in Bulls and Cows.

|  |
| --- |
| 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 |

The RedrawAt override erases the old sprite by overwriting it with spaces, then writing the new sprite in the new position on screen.

|  |
| --- |
| End Class |

**3.6.6: Base Class - Tick Entity**

|  |
| --- |
| 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 |

A TickEntity is a sprite entity which has a handler for each in-game tick (update instant). This is the base class for IteratingSpriteEntity and GravityEntity. The constructor has the same parameters as the base class SpriteEntity, but also adds a tick handler to the global Tick event.

|  |
| --- |
| Public Overrides Sub Dispose()  RemoveHandler Tick, AddressOf OnTick  MyBase.Dispose()  End Sub |

The Dispose override removes the tick handler to avoid the object being kept alive due to it being referenced in the global Tick event and not garbage collected, which unnecessarily consumes memory.

|  |
| --- |
| End Class |

**3.6.7: Iterating Sprite Entity**

|  |
| --- |
| 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 |

An IteratingSpriteEntity is a tick entity which tick handler iterates its sprites. The constructor requires a collection of entities and some sprites for iterating through each tick. For example, the fireworks in the Win Screen are all instances of IteratingSpriteEntity.

|  |
| --- |
| Protected Overrides Sub OnTick()  If sprites.MoveNext() Then Sprite = sprites.Current Else Dispose()  End Sub |

The OnTick override iterates the sprites each tick.

|  |
| --- |
| Public Overrides Sub Dispose()  sprites.Dispose()  MyBase.Dispose()  End Sub  End Class |

The Dispose override also disposes the iterator as it may have underlying memory to release.

**3.6.8: Gravity Entity**

|  |
| --- |
| Class GravityEntity  Inherits TickEntity  Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)  MyBase.New(entities, sprite)  End Sub |

A GravityEntity is a tick entity which tick handler simulates gravity. This is the base class for PlayerEntity and GravityEntityFactoryEntity. The constructor has the same parameters as the base class TickEntity.

|  |
| --- |
| 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 |

The OnTick override moves the entity upwards and decreases its vertical velocity if there is positive vertical velocity. If not, if no other entity supports it below, the entity is moved downwards, simulating gravity.

|  |
| --- |
| Public Event GroundHit() |

The GroundHit event is raised whenever the GravityEntity hits the ground. To avoid repeatedly calling the handlers, all handlers are removed after this event is raised every time.

|  |
| --- |
| 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 |

The GoUp override does nothing when the entity is in mid-air. If not, it gives the entity vertical velocity. This simulates jumping.

|  |
| --- |
| End Class |

**3.6.9: Player Entity**

|  |
| --- |
| 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 |

A PlayerEntity is a gravity entity controlled by the player. An alternate name for this is an “avatar”, also used in previous text. The constructor has the same parameters as the base class GravityEntity.

|  |
| --- |
| 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 |

The HandleKey method handles arrow keys inputted by the player and calls one of the four Go methods depending on the arrow key inputted, if the entity is not inside a TriggerZone currently or the current TriggerZone has not disabled the arrow keys. It is called by the main program which handles ticking.

|  |
| --- |
| 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 |

The GoLeft override teleports the avatar to the right window edge of the previous zone if it is in contact with the left window edge.

|  |
| --- |
| 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 |

The GoRight override teleports the avatar to the left window edge of the next zone if it is in contact with the left window edge, saving the new progress inside CurrentRegion.GoRight().

|  |
| --- |
| End Class |

**3.6.10: Gravity Entity Factory**

|  |
| --- |
| Class GravityEntityFactory |

A GravityEntityFactory is a generator, akin to a factory, of instances of GravityEntityFactoryEntity, which are just instances of GravityEntity but owned by a GravityEntityFactory. The factory stores a record of the factory entities it has produced, and has the power to dispose of them all at once. As an example of where this is used, the pieces in Connect Four are instances of GravityEntityFactoryEntity.

|  |
| --- |
| 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 |

The GravityEntityFactoryEntity constructor requires a collection of entities, a sprite, a GravityEntityFactory as owner, an initial position and a handler for when the entity hits the ground.

|  |
| --- |
| Public Sub New(entities As ICollection(Of Entity), sprite As Sprite)  Me.Entities = entities  Me.Sprite = sprite  End Sub |

The GravityEntityFactory constructor requires a collection of entities and a sprite as parameters.

|  |
| --- |
| ReadOnly Sprite As Sprite  ReadOnly Entities As ICollection(Of Entity)  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 |

The Add method takes an initial position and optionally a handler called when the ground is hit, then generating a GravityEntityFactoryEntity which is immediately displayed if position is not null.

|  |
| --- |
| 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 |

The Clear method disposes of all instances of GravityEntityFactoryEntity belonging to this factory.

|  |
| --- |
| 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 |

The ItemAt method returns the GravityEntityFactoryEntity belonging to this factory occupying the provided location if found.

|  |
| --- |
| End Class |

**3.7: Implementation of sprites**

Below are the sprites available.

|  |
| --- |
| 'Unicode:  '1.1☺☹☠❣❤✌☝✍♨✈⌛⌚☀☁☂❄☃☄♠♥♦♣♟☎⌨✉✏✒✂☢☣↗➡↘↙↖↕↔↩↪✡☸☯✝☦☪☮♈♉♊♋♌♍♎♏♐♑♒♓▶◀♀♂☑✔✖✳✴❇‼〰©®™Ⓜ  '1.1㊗㊙▪▫☜♅♪♜☌♘☛♞☵☒♛♢✎‍♡☼☴♆☲☇♇☏☨☧☤☥♭☭☽☾❥☍☋☊☬♧☉#☞☶♁♤☷✐♮♖★♝\*☰☫♫♙♃☚♬☩♄☓♯☟☈☻☱♕☳♔♩♚♗☡☐  '3.0⁉♱♰☙  '3.2⤴⤵♻〽◼◻◾◽☖♷⚁⚄⚆⚈♼☗♵⚉⚀⚇♹♲♸⚂♺♴⚅♳♽⚃♶  '4.0☕☔⚡⚠⬆⬇⬅⏏⚏⚋⚎⚑⚊⚍⚐⚌  '4.1☘⚓⚒⚔⚙⚖⚗⚰⚱♿⚛⚕♾⚜⚫⚪⚩⚭⚢⚥⚘⚤⚦⚨⚣⚬⚮⚚⚯⚧  '5.0 // ⚲  '5.1⭐⬛⬜⚶⚼⚸⚴⚹⚳⚵⚻⚷⚝⚺ // ⛂⛁⛃⛀  '5.2⛪⛲⛺⛽⛵⛅⛄⚽⚾⛳’⛔⭕❗ // ⛩⛴⛈⛱⛸⛑⛏⛓⛷⛹⛰⛟⛙⛞⛮⛶⛯⛜⛡⛿⛣⛊⛐⛾⛉⛚⛘⛠⛆⛝⛌⛕⛬⛍⛫⛖⚞⛨⚟⛻⛋⛒⛛⛭⛇⛼⚿⛗  '6.0✋✊⏳⏰✨⛎⏩⏪⏫⏬✅❌❎➕➖➗➰➿❓❔❕ // ⏱⏲⏭⏯⏮⛧⛢⛤ // Right-Handed interlaced pentagram: ⛥ Left-Handed interlaced pentagram: ⛦  '7.0 // ⏸⏹⏺  '10. // ₿ |

The above are the emoji characters that display correctly in a Windows 7 CLI. It can be seen that a large amount of emojis have been filtered out from Unicode. Only Unicode 1.1 emojis have been used in this program as they are the most widely supported characters – even in Microsoft Word!

|  |
| --- |
| Const Empty = " "c  ReadOnly Empty\_ As New Sprite(Empty)  ReadOnly GlobalEntities As New HashSet(Of Entity) |

The Empty constants enable easy modification of code. The GlobalEntities represent entities that persist across rooms – like the player entity. In contrast to “local entities” (Entities of Regions), which are removed once the associated region is left by the player, the global entities persist throughout the whole program.

|  |
| --- |
| Public ReadOnly Sunny\_ As New Sprite("☺"c, ConsoleColor.Green)  Public ReadOnly Sunny\_Angry As New Sprite("☹"c, ConsoleColor.Red)  Public ReadOnly Sunny As New PlayerEntity(GlobalEntities, Sunny\_) |

The Sunny sprites are the possible appearances of Sunny the player entity, a green one in a normal and happy mood, and a red one in a sad and angry mood. From now on, the player entity or the avatar will be referred as “Sunny” – the protagonist of this program – “SBA: Sunny’s Big Adventure”.

|  |
| --- |
| Public ReadOnly Sun\_ As New Sprite("☼"c, ConsoleColor.Yellow)  Public ReadOnly Sun As New SpriteEntity(GlobalEntities, Sun\_)  Public ReadOnly Horsey\_ As New Sprite("♘"c, ConsoleColor.Magenta)  Public ReadOnly Horsey\_Dead As New Sprite("♞"c, ConsoleColor.DarkMagenta)  Public ReadOnly Horsey As New SpriteEntity(GlobalEntities, Horsey\_) |

These sprites and entities have remained unused. The Sun was originally intended as a prized possession of Sunny, but got stolen by the antagonist Horsey. This was the storyline of this program; however, it was not implemented due to time constraints.

|  |
| --- |
| Public ReadOnly Firework\_1 As New Sprite("✳"c, ConsoleColor.Yellow)  Public ReadOnly Firework\_2 As New Sprite("✴"c, ConsoleColor.Red)  Public ReadOnly Firework\_3 As New Sprite("❇"c, ConsoleColor.DarkRed) |

These are the three sprites the fireworks in the Win Screen go through. The first is a small yellow firework, the second is a red bright firework, and the last is a dark red dim firework. This makes the fireworks more lively and rewarding for the player.

|  |
| --- |
| Public ActiveEntity As PlayerEntity = Sunny |

This program was originally intended to have a multiple playable characters: multiple player entities at once, achieved by an ActiveEntity property as the current controlling entity, but that was deemed too time-consuming and the idea was scrapped.

**3.8: Implementation of regions (Process implementation)**

Implementation of each the game processes are contained within “regions” which act as rooms for 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.

**3.8.1: Base Class - Region**

This is the common base class of all regions. This part will explain the details inside this class.

|  |
| --- |
| Dim \_currentRegion As Region  MustInherit Class Region  Implements IDisposable  Sub New(Optional bedrock As Boolean = True)  If bedrock Then Equals(New RectangleEntity(WriteEntities,  New Rectangle(0, WindowHeight - 2, WindowWidth, 1)), Nothing)  End Sub |

The constructor accepts an optional flag to disable the region bedrock. This is unused inside the program.

|  |
| --- |
| ''' <returns>Whether region was changed.</returns>  Public Function GoLeft() As Boolean  If Left IsNot Nothing Then  SetCurrentRegion = Left  Return True  End If  Return False  End Function  ''' <returns>Whether region was changed.</returns>  Public Function GoRight() As Boolean  If Right IsNot Nothing Then  SetCurrentRegion = Right  SaveRegion(CurrentRegion)  Return True  End If  Return False  End Function  Protected MustOverride ReadOnly Property Left As Func(Of Region)  Protected MustOverride ReadOnly Property Right As Func(Of Region) |

The regions to the left and the right of the current region are lazily constructed via region-returning functions. This is to avoid infinite recursion during object construction.

|  |
| --- |
| Protected ReadOnly WriteEntities As New List(Of Entity)(GlobalEntities)  Public ReadOnly Entities As New ReadOnlyCollection(Of Entity)(WriteEntities) |

The list of entities is not modifiable outside this region to avoid external interference causing unexpected entities spawning on the screen.

|  |
| --- |
| Public Sub Dispose() Implements IDisposable.Dispose  For Each entity In Entities.Except(GlobalEntities)  entity.Dispose()  Next  End Sub |

Whenever a region is disposed, all local entities are disposed to avoid objects lingering between regions.

|  |
| --- |
| End Class  Public ReadOnly Property CurrentRegion As Region  Get  Return \_currentRegion  End Get  End Property  Public WriteOnly Property SetCurrentRegion As Func(Of Region)  Set(value As Func(Of Region))  \_currentRegion.Dispose()  ' \*\*Prevent collision detection between old and new region entities\*\*  \_currentRegion = value()  End Set  End Property |

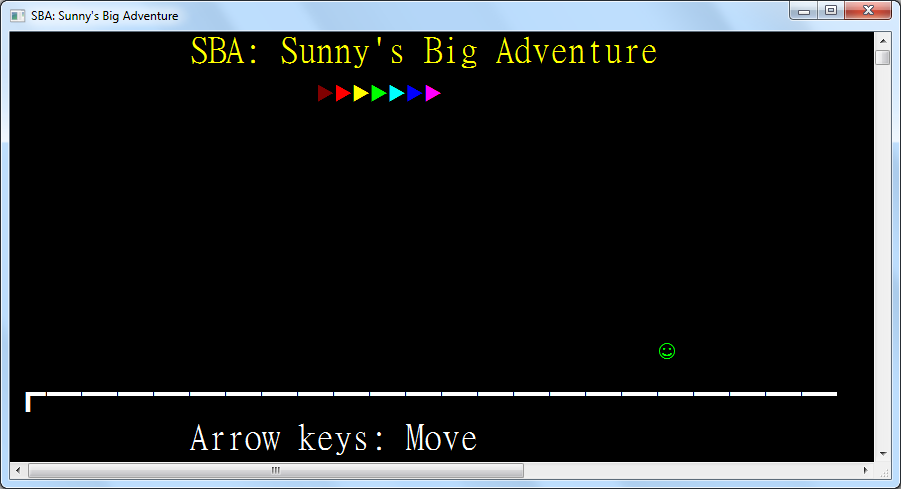
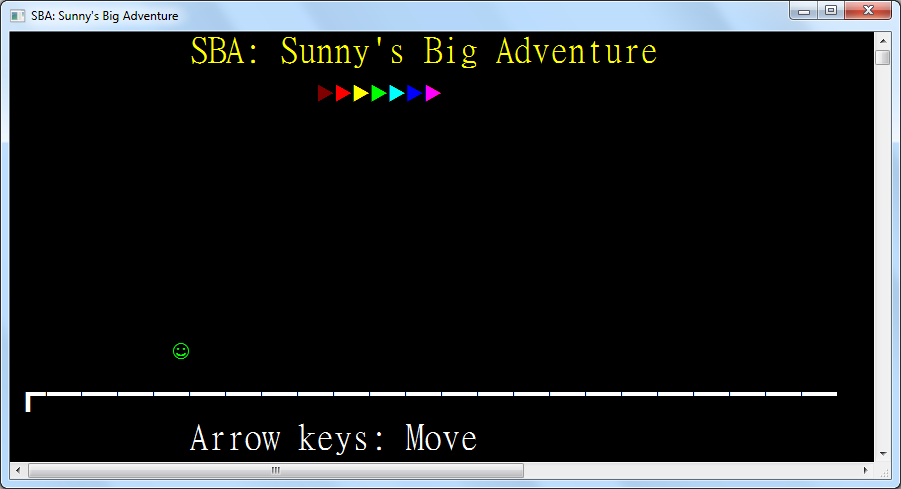
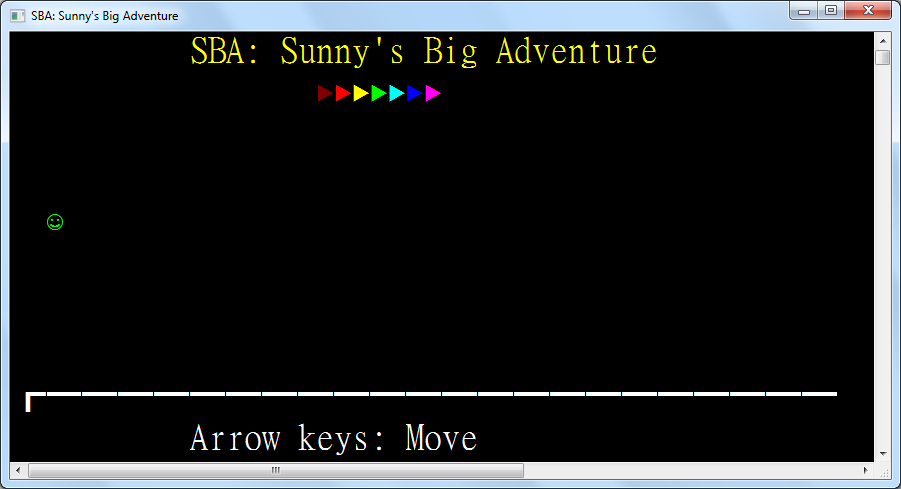
Before a new region is set, the previous region needs to be disposed. This is the reason why lazy construction is needed when changing regions.

**3.8.2: Region 1 – Title Screen**

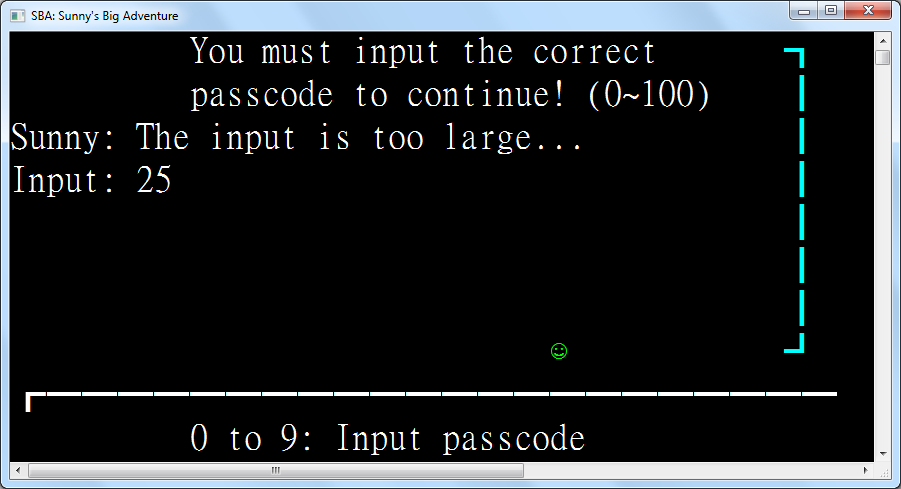


This is the title screen where the player is greeted with rainbow-colored arrows as well as Sunny with a happy face. The player will need to press the right arrow key to move Sunny to the right to continue to the next region.

|  |
| --- |
| Class Region1\_Title  Inherits Region  Protected ReadOnly SBA As New TextEntity(WriteEntities, "SBA: Sunny's Big Adventure", New Point(10, 0), ConsoleColor.Yellow)  Protected ReadOnly Arrow1 As New TextEntity(WriteEntities, "▶", New Point(17, 1), ConsoleColor.DarkRed)  Protected ReadOnly Arrow2 As New TextEntity(WriteEntities, "▶", New Point(18, 1), ConsoleColor.Red)  Protected ReadOnly Arrow3 As New TextEntity(WriteEntities, "▶", New Point(19, 1), ConsoleColor.Yellow)  Protected ReadOnly Arrow4 As New TextEntity(WriteEntities, "▶", New Point(20, 1), ConsoleColor.Green)  Protected ReadOnly Arrow5 As New TextEntity(WriteEntities, "▶", New Point(21, 1), ConsoleColor.Cyan)  Protected ReadOnly Arrow6 As New TextEntity(WriteEntities, "▶", New Point(22, 1), ConsoleColor.Blue)  Protected ReadOnly Arrow7 As New TextEntity(WriteEntities, "▶", New Point(23, 1), ConsoleColor.Magenta)  Protected ReadOnly Keybinds As New TextEntity(WriteEntities, "Arrow keys: Move", New Point(10, 9))  Protected ReadOnly Trigger As New TriggerZone(WriteEntities, New Rectangle(30, 1, WindowWidth - 30, 8))  Protected Overrides ReadOnly Property Left As Func(Of Region) = Nothing  Protected Overrides ReadOnly Property Right As Func(Of Region) = Function() New Region2\_NumberGuess()  End Class |



**3.8.3: Region 2 – Number Guess**



This is the number guessing game where the player presses the numeric keys, the backspace key and the enter key to guess a number between 0 and 100. The game will respond by telling if the input is too large or small, which will make Sunny sad and angry, or when the input is correct, the barrier will be removed.

|  |
| --- |
| Class Region2\_NumberGuess  Inherits Region  Protected Passcode As Byte = CByte(Random.Next(101))  Protected ReadOnly Instruction As New TextEntity(WriteEntities, "You must input the correct")  Protected ReadOnly Instruction2 As New TextEntity(WriteEntities, "passcode to continue! (0~100)")  Protected ReadOnly Instruction3 As New TextEntity(WriteEntities, "Sunny: I must guess it...")  Protected ReadOnly Instruction4 As New TextEntity(WriteEntities, "0 to 9: Input passcode")  Protected ReadOnly Input As New TextEntity(WriteEntities, "Input: ") |

These text entities are not given initial coordinates since they appear in sequence after the player approaches the barrier, facilitated by trigger zones, simulating an adventurous environment.

|  |
| --- |
| Protected ReadOnly Barrier As New RectangleEntity(WriteEntities, New Rectangle(42, 0, 2, 8), ConsoleColor.Cyan)  Protected ReadOnly Trigger As New TriggerZone(WriteEntities, New Rectangle(30, 6, 6, 3),  Function(key)  ActiveEntity.Sprite = Sunny\_  Select Case key  Case ConsoleKey.D0 To ConsoleKey.D9  Input.Text &= key.ToString()(1) |

The number input is concatenated with the pressed numeric key to behave as numeric input.

|  |
| --- |
| Case ConsoleKey.Backspace  If Input.Text <> "Input: " Then \_  Input.Text = Input.Text.Substring(0, Input.Text.Length - 1) |

The backspace removes the last digit of the number input until all digits have been removed.

|  |
| --- |
| Case ConsoleKey.Enter  Dim inputNumber As Byte  If Byte.TryParse(String.Concat(Input.Text.SkipWhile(Function(c) Not Char.IsDigit(c))), inputNumber) Then |

When the enter key is pressed, the number input is first parsed as a byte which has a range of 0 to 255, perfect for parsing a number between 0 and 100. Since Input also contains "Input: ", non-digit characters are skipped.

|  |
| --- |
| Select Case inputNumber  Case Is < Passcode  Instruction3.Text = "Sunny: The input is too small..."  ActiveEntity.Sprite = Sunny\_Angry  Case Is > Passcode  Instruction3.Text = "Sunny: The input is too large..."  ActiveEntity.Sprite = Sunny\_Angry |

If the number is too small or too large, Sunny becomes sad and angry.

|  |
| --- |
| Case Else  Instruction3.Color = ConsoleColor.Green  Instruction3.Text = "Sunny: Yes! The passcode is correct!"  Instruction.Dispose()  Instruction2.Dispose()  Input.Dispose()  Barrier.Dispose()  Trigger.Dispose() |

If the number is correct, the now-outdated input and instruction text are disposed. The barrier blocking the path and the trigger zone disabling arrow keys and detecting numeric input are disposed as well.

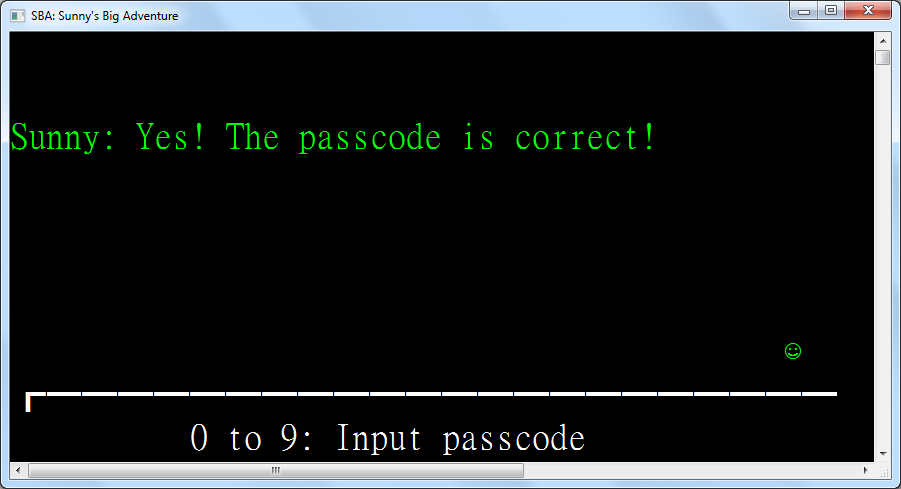
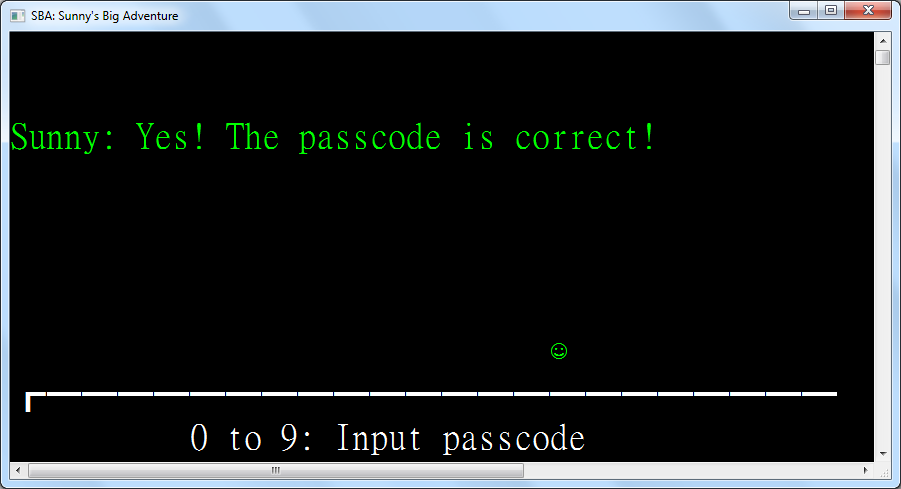
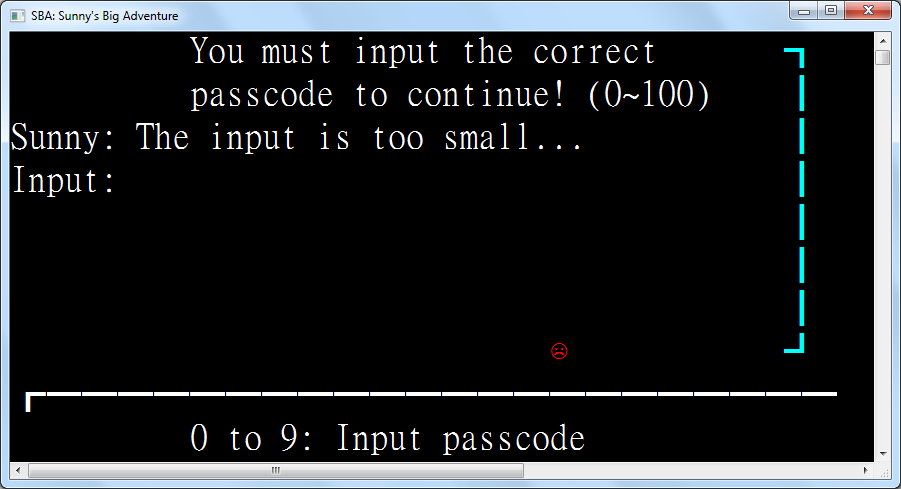
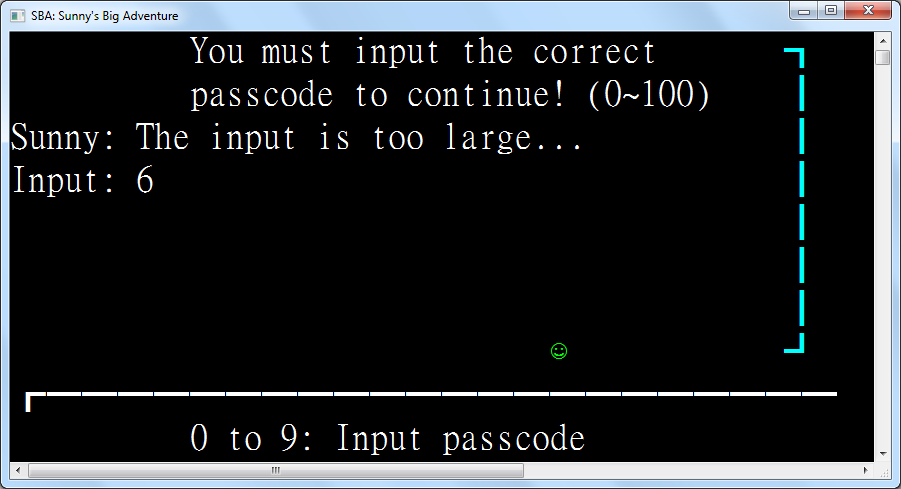
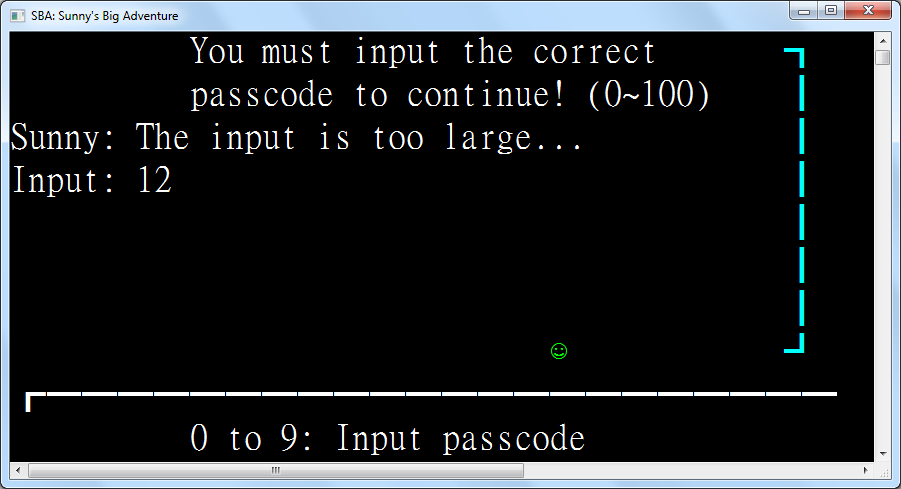
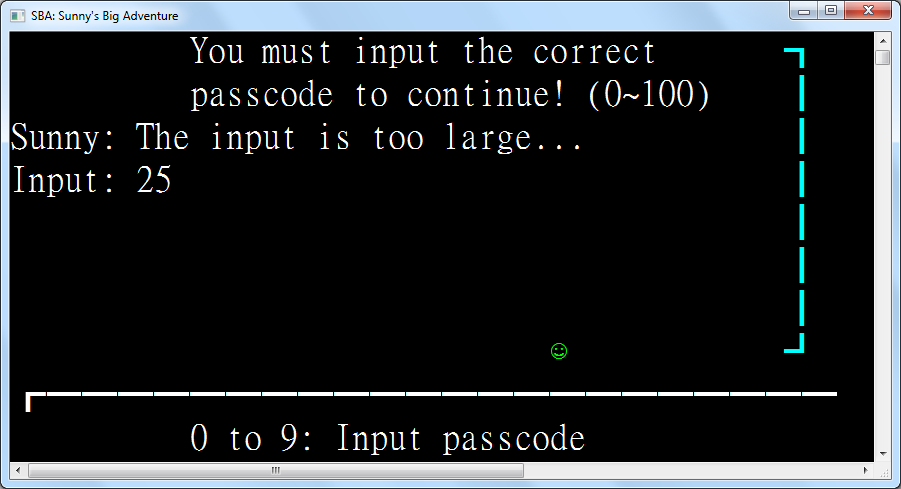
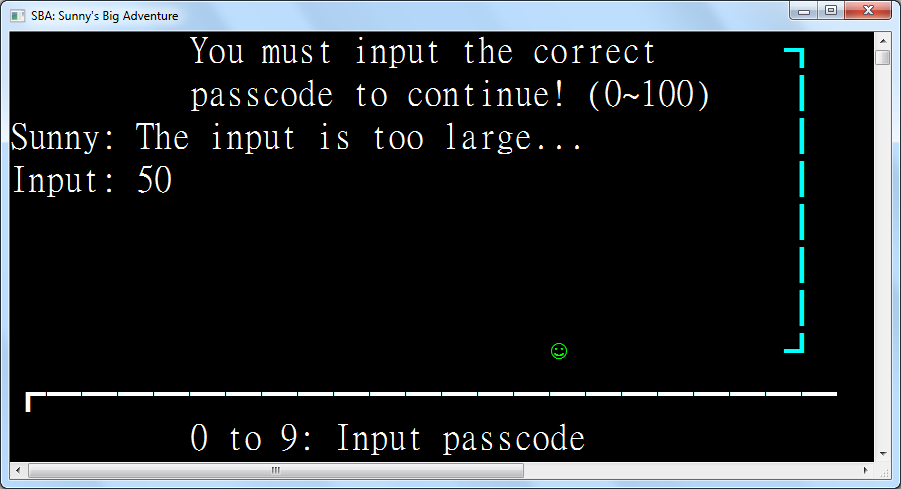
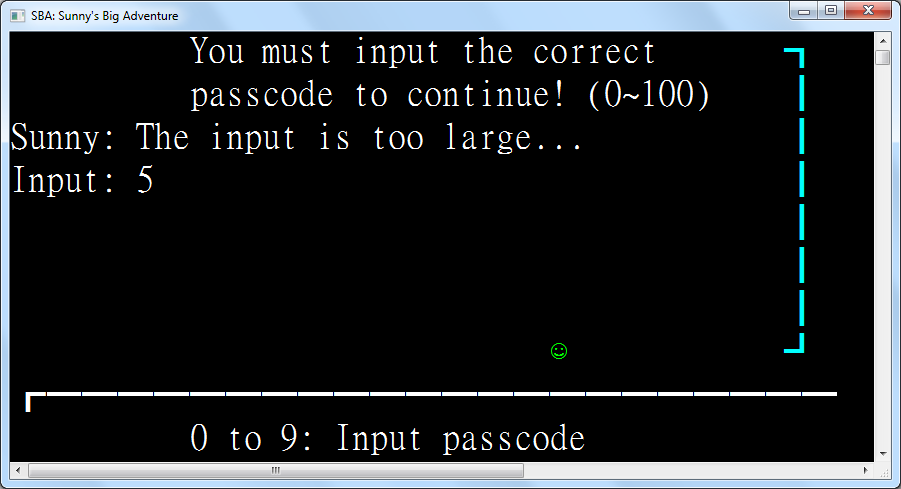
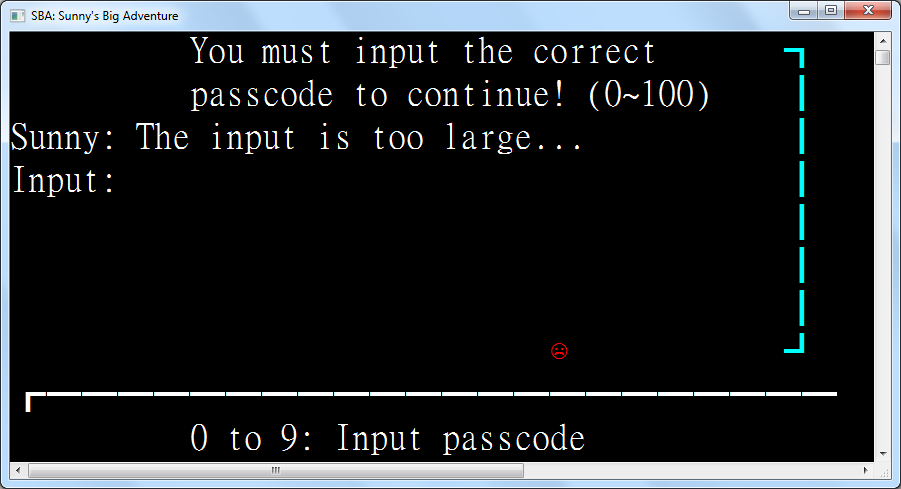
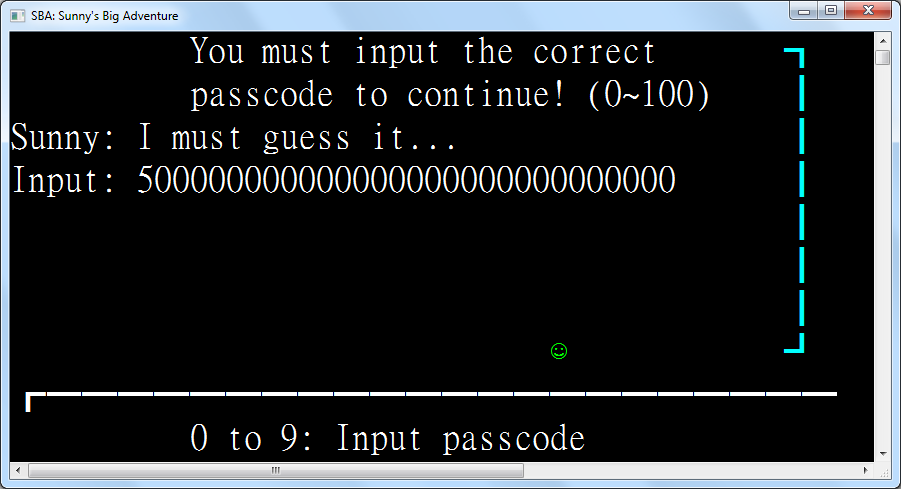
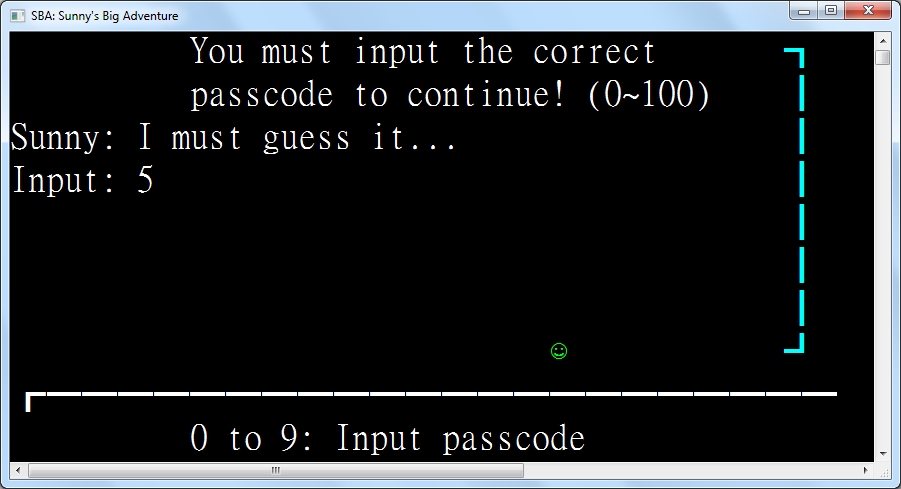
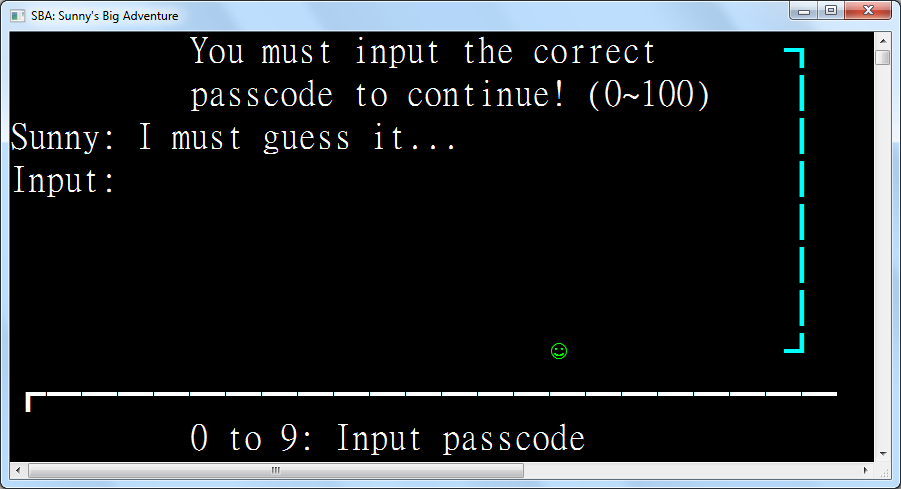
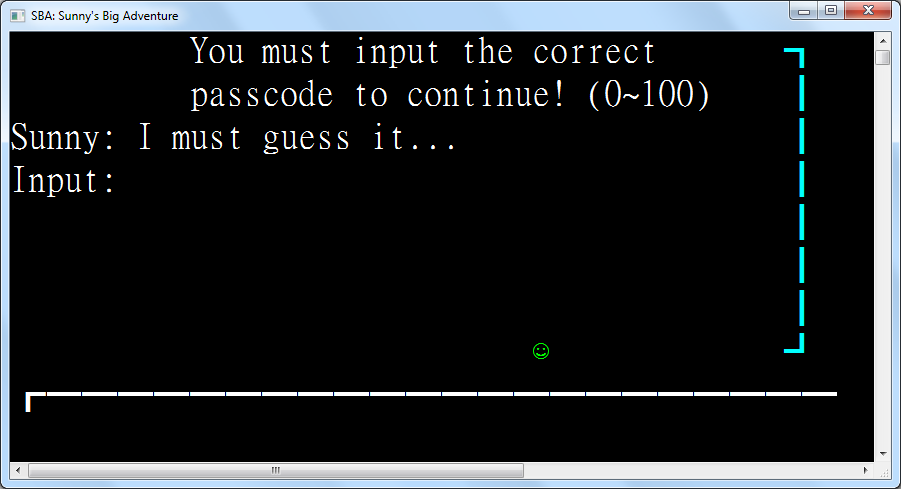
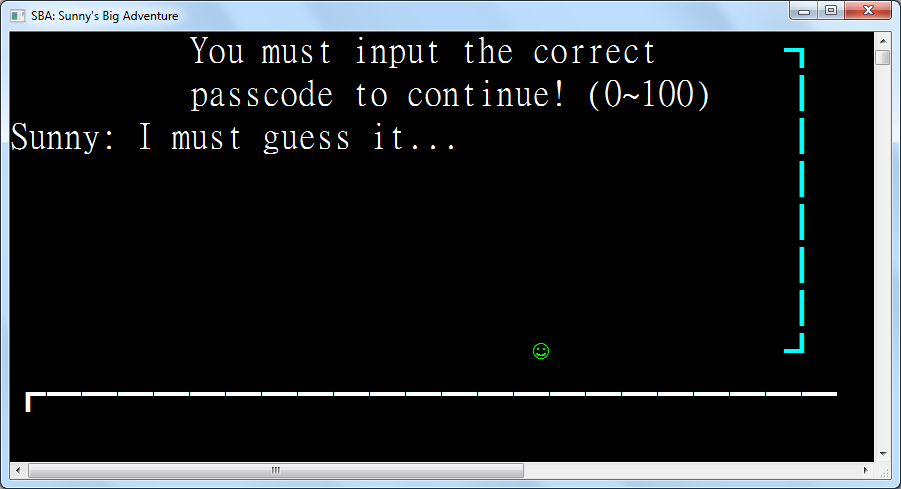
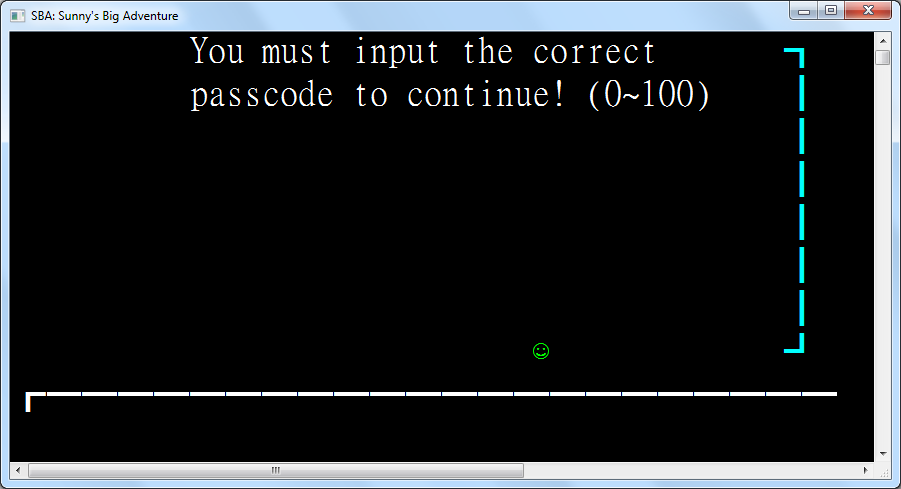
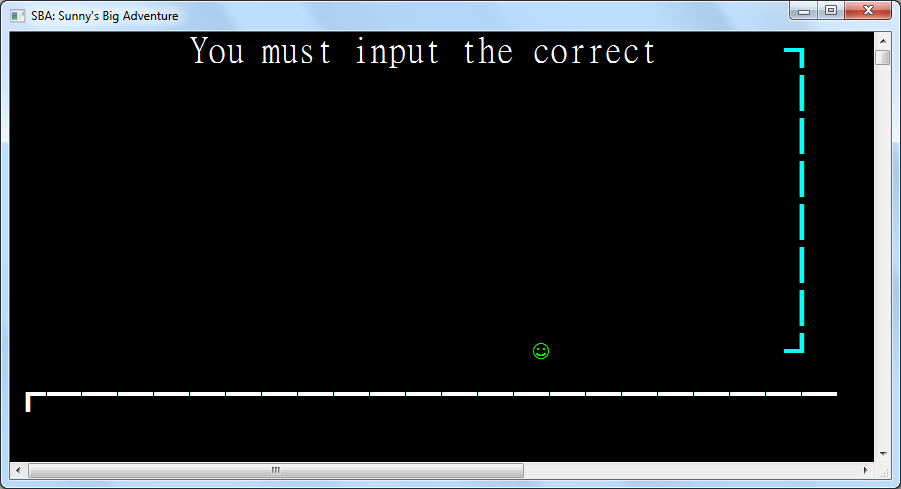
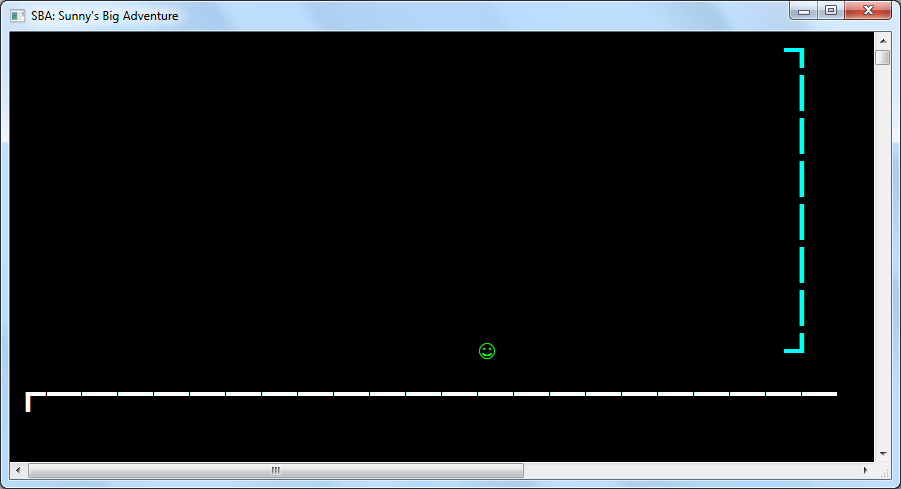
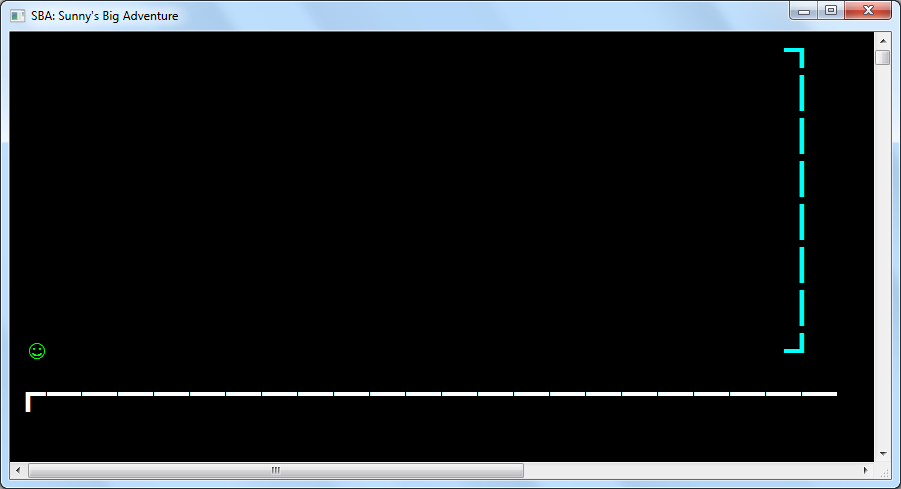
|  |
| --- |
| End Select  Else  Instruction3.Text = "Sunny: The input is too large..." ' inputNumber > 255 |

If the number fails to be parsed, it is because it exceeds the byte range, 0 to 255, which is obviously too large.

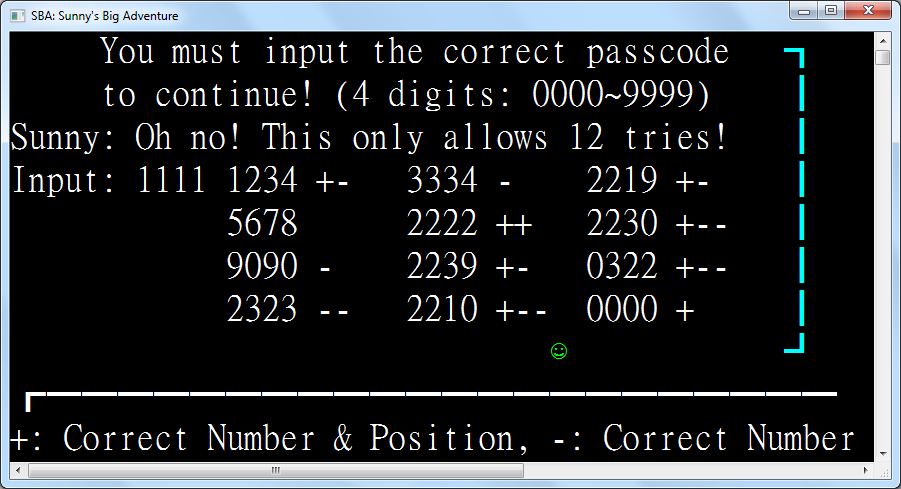
|  |
| --- |
| End If  Input.Text = "Input: "  End Select  Return True  End Function,  Sub()  Instruction.Position = New Point(10, 0)  Threading.Thread.Sleep(500)  Instruction2.Position = New Point(10, 1)  Threading.Thread.Sleep(500)  Instruction3.Position = New Point(0, 2)  Threading.Thread.Sleep(500)  Input.Position = New Point(0, 3)  Threading.Thread.Sleep(500)  Instruction4.Position = New Point(10, 9) |

When Sunny enters the trigger zone in front of the barrier, the instruction texts appear sequentially with a 0.5s delay between each texts’ appearance on screen.

|  |
| --- |
| End Sub)  Protected Overrides ReadOnly Property Left As Func(Of Region) = Function() New Region1\_Title()  Protected Overrides ReadOnly Property Right As Func(Of Region) = Function() New Region3\_BullsAndCows()  End Class |



**3.8.4: Region 3 – Bulls and Cows**



|  |
| --- |
| Class Region3\_BullsAndCows  Inherits Region  Protected Passcode As Short = CShort(Random.Next(10000)) |

A random passcode is generated whenever this room is entered. A number between 0 and 9999 is generated.

|  |
| --- |
| Protected ReadOnly Instruction As New TextEntity(WriteEntities, "You must input the correct passcode")  Protected ReadOnly Instruction2 As New TextEntity(WriteEntities, "to continue! (4 digits: 0000~9999)")  Protected ReadOnly Instruction3 As New TextEntity(WriteEntities, "Sunny: Oh no! This only allows 12 tries!")  Protected ReadOnly Instruction4 As New TextEntity(WriteEntities, "+: Correct Number & Position, -: Correct Number")  Protected ReadOnly Input As New TextEntity(WriteEntities, "Input: ") |

The instructions for this game will appear sequentially when Sunny approaches the barrier so they are initially invisible. This game allows only 12 tries, limited by the available drawing area, so the player is reminded of this.

|  |
| --- |
| Protected ReadOnly Barrier As New RectangleEntity(WriteEntities, New Rectangle(42, 0, 2, 8), ConsoleColor.Cyan)  Protected ReadOnly Trigger As New TriggerZone(WriteEntities, New Rectangle(30, 6, 6, 3),  Function(key)  ActiveEntity.Sprite = Sunny\_  Select Case key  Case ConsoleKey.D0 To ConsoleKey.D9  If Input.Text.Length < "Input: 9999".Length Then Input.Text &= key.ToString()(1) |

When a numeric key is entered and less than 4 digits are entered, the corresponding digit will be inputted.

|  |
| --- |
| Case ConsoleKey.Backspace  If Input.Text <> "Input: " Then \_  Input.Text = Input.Text.Substring(0, Input.Text.Length - 1) |

When the backspace is pressed and there are digits to remove, the last digit is removed from the input.

|  |
| --- |
| Case ConsoleKey.Enter  If Input.Text.Length = "Input: 9999".Length Then  Dim inputNumber = Short.Parse(String.Concat(  Input.Text.SkipWhile(Function(c) Not Char.IsDigit(c)))) |

As the input text is prefixed by "Input: ", the number is extracted after skipping the prefix non-digits.

|  |
| --- |
| Dim t = Matches(inputNumber, Passcode)  Dim p = NextPosition() |

These two functions Matches and NextPosition, are defined below.

|  |
| --- |
| If t.CorrectNumPos = 4 Then  Input.Color = ConsoleColor.Green  Input.Text = "Gate open!"  Barrier.Dispose()  Trigger.Dispose() |

If all 4 digits inputted matches the random number in both numeric value and digit position, the gate is opened.

|  |
| --- |
| ElseIf p.HasValue Then  Input.Text = "Input: "  Dim e As New TextEntity(WriteEntities, String.Concat(inputNumber.ToString().PadLeft(4, "0"c),  " ", New String("+"c, t.CorrectNumPos), New String("-"c, t.CorrectNum)), p)  ActiveEntity.Sprite = Sunny\_Angry |

If the digits do not match the random number and the 12-try limit is not reached, the input is cleared, and then pluses, indicating that the correct numeric value and digit position is guessed, followed by minuses, indicating the correct numeric value but incorrect digit position, are displayed.

|  |
| --- |
| Else  Input.Color = ConsoleColor.Red  Input.Text = "Gate locked."  ActiveEntity.Sprite = Sunny\_Angry  Trigger.Dispose()  End If |

If the 12-try limit is reached, the gate is locked and Sunny becomes angry. The room will need to be re-entered to retry the whole game with a different random number.

|  |
| --- |
| Else  Input.Text = "Input: " |

If the input number is less than 4 digits long, the input will be cleared and nothing will occur.

|  |
| --- |
| End If  End Select  Return True  End Function,  Sub()  ActiveEntity.Sprite = Sunny\_  Instruction.Position = New Point(5, 0)  Threading.Thread.Sleep(500)  Instruction2.Position = New Point(5, 1)  Threading.Thread.Sleep(500)  Instruction3.Position = New Point(0, 2)  Threading.Thread.Sleep(500)  Input.Position = New Point(0, 3)  Threading.Thread.Sleep(500)  Instruction4.Position = New Point(0, 9) |

The instructions are shown sequentially with a 0.5s delay in-between when Sunny approaches the gate.

|  |
| --- |
| End Sub)  Public Shared Function Matches(guess As Short, actual As Short) As (CorrectNumPos As Integer, CorrectNum As Integer)  Dim g = guess.ToString().PadLeft(4, "0"c)  Dim a = actual.ToString().PadLeft(4, "0"c) |

When the number is interpreted as a string of digits, the number will be stripped of leading zeroes by default. Before comparing the guess and actual numbers, they are left-padded with zeroes until they are 4 digits long.

|  |
| --- |
| Dim correctNumPosCount = g.Zip(a, Function(gc, ac) gc = ac).Count(Function(b) b) |

The number of digits with correct value and position can be obtained by comparing each digit of the two numbers sequentially.

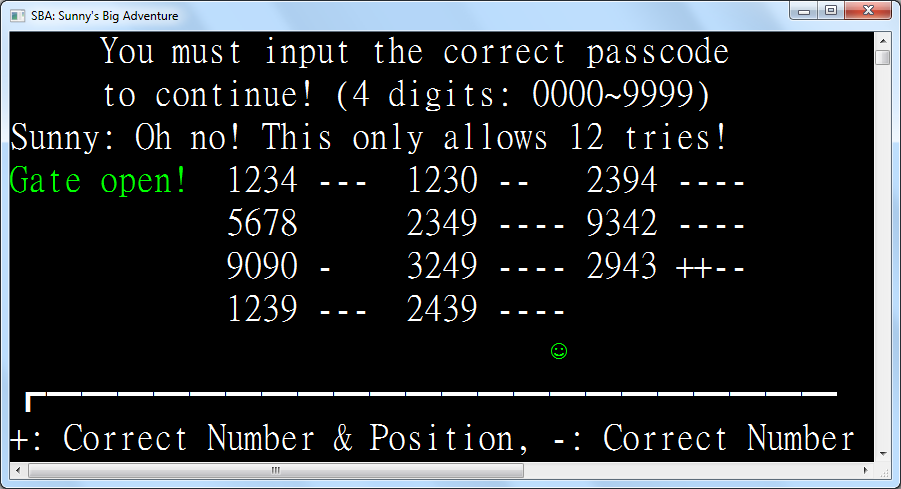
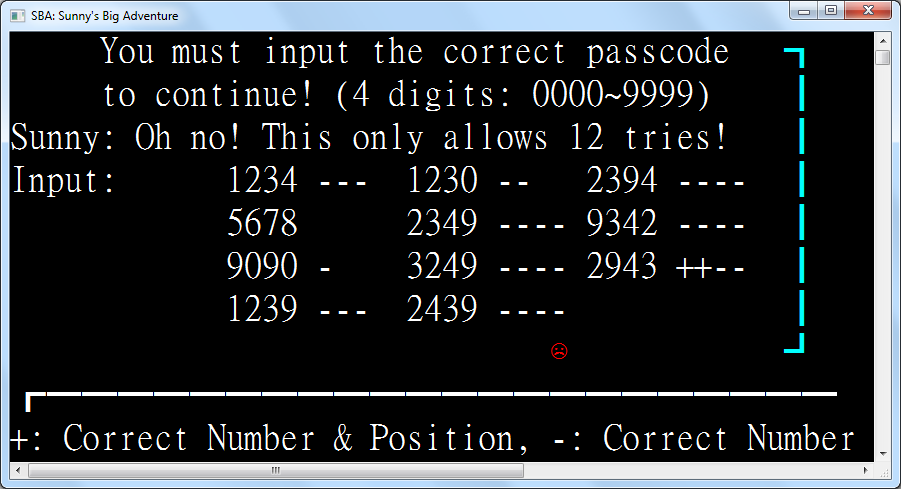
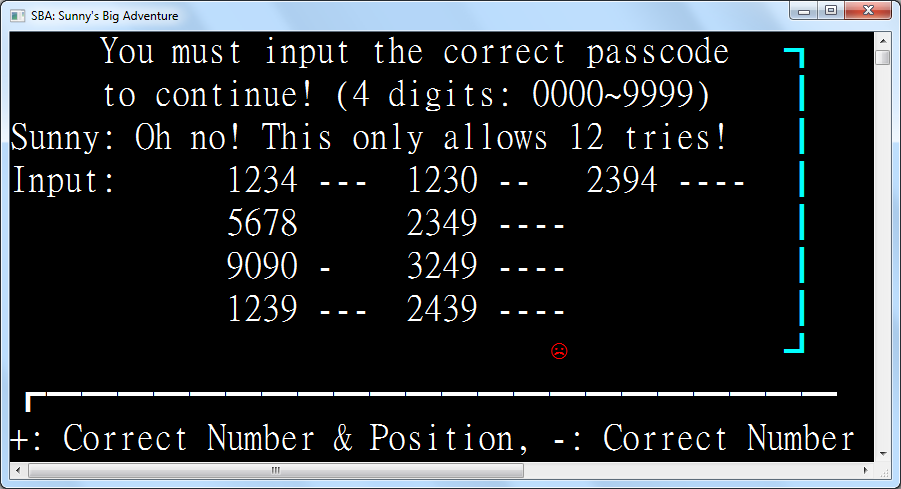
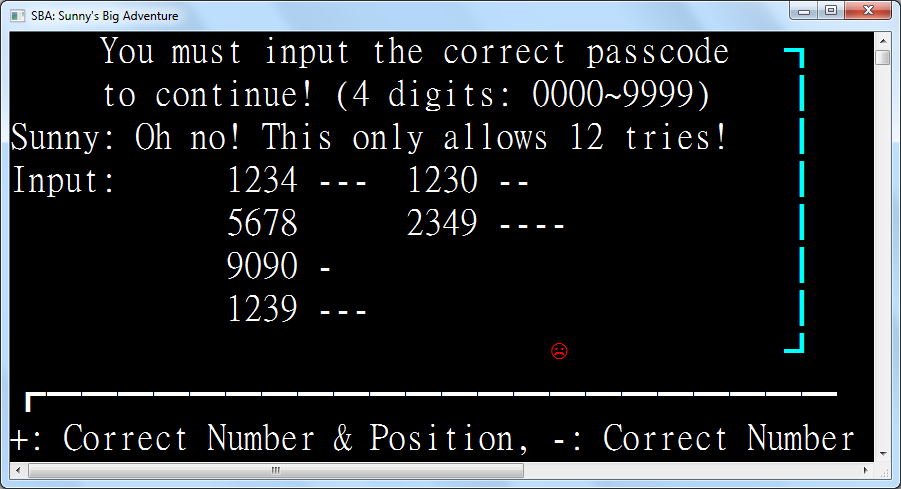
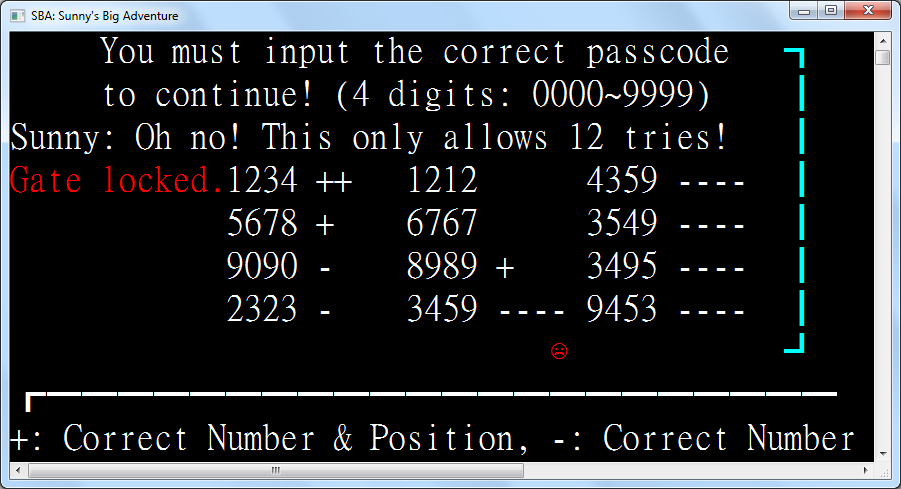
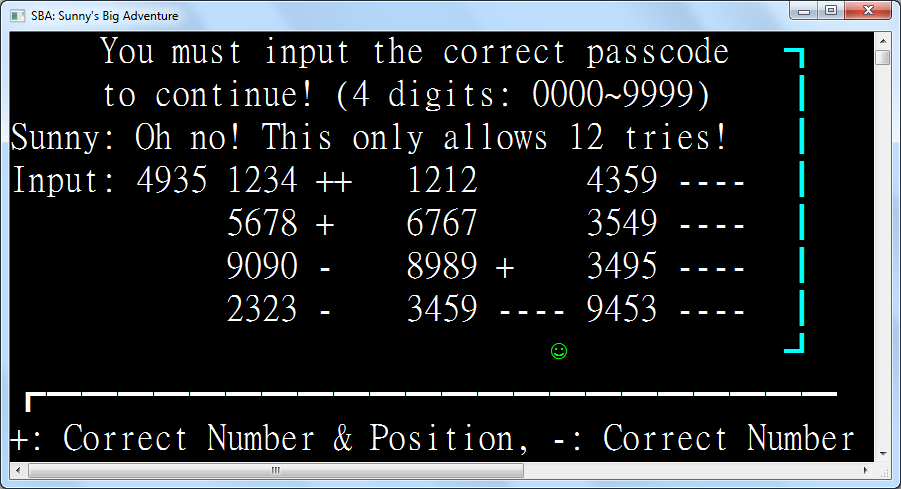
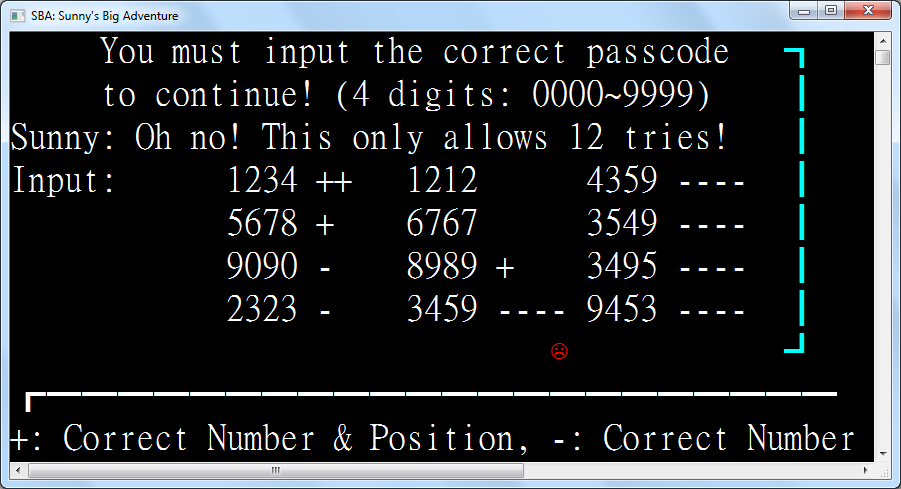
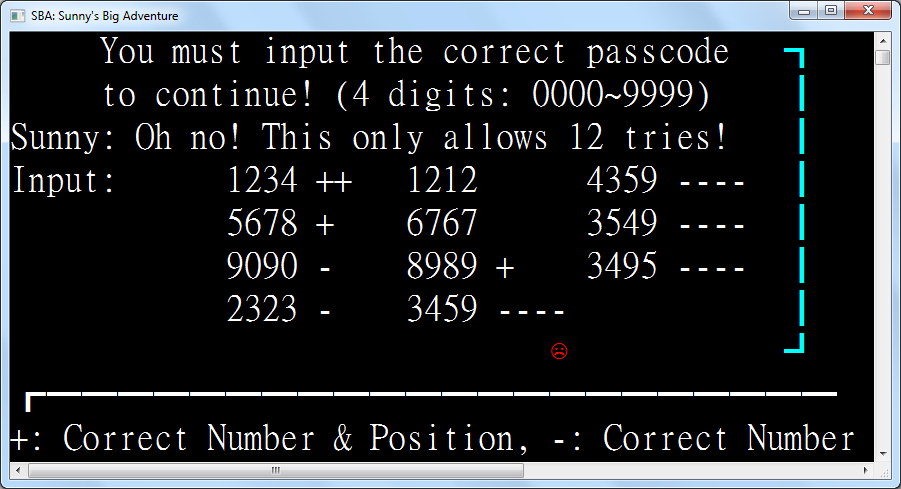
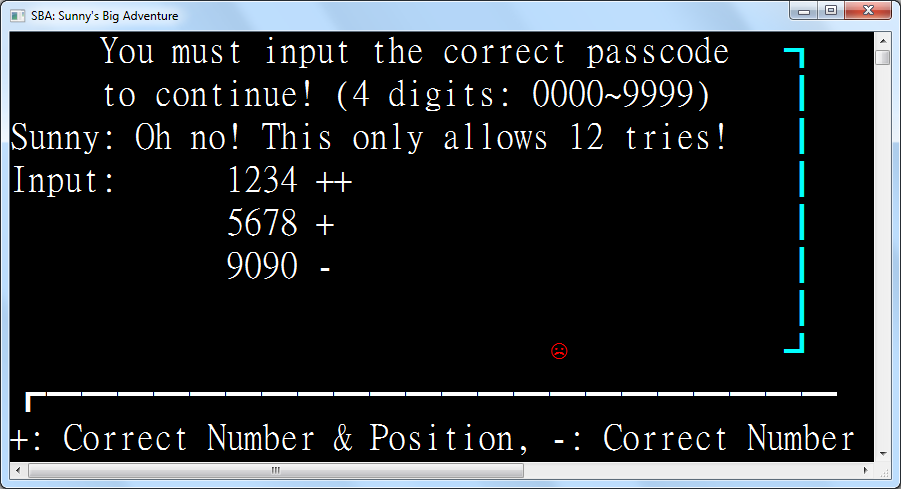
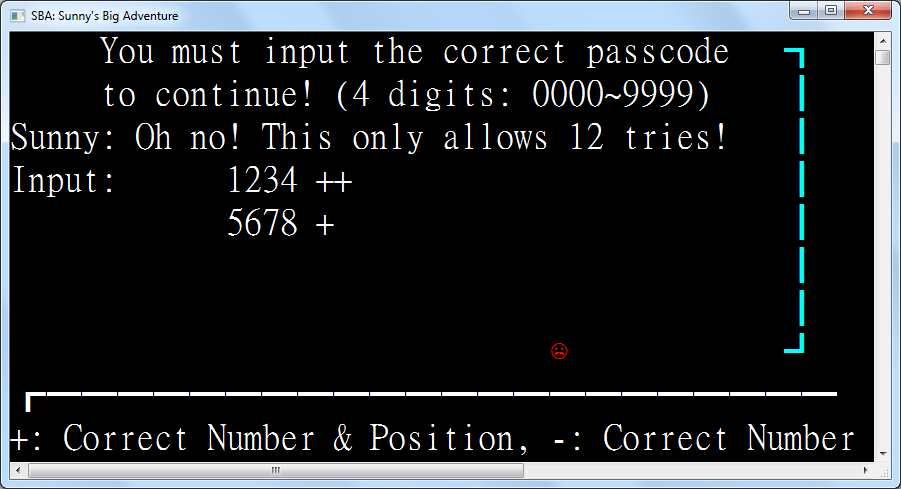
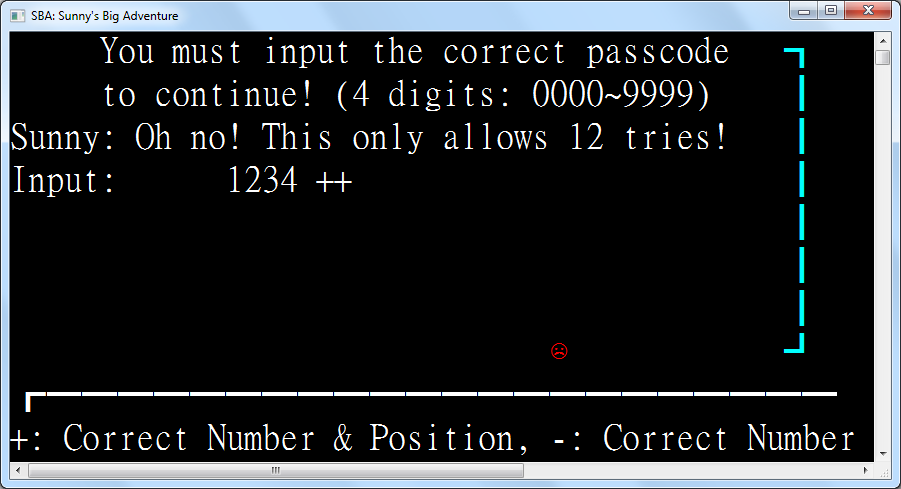
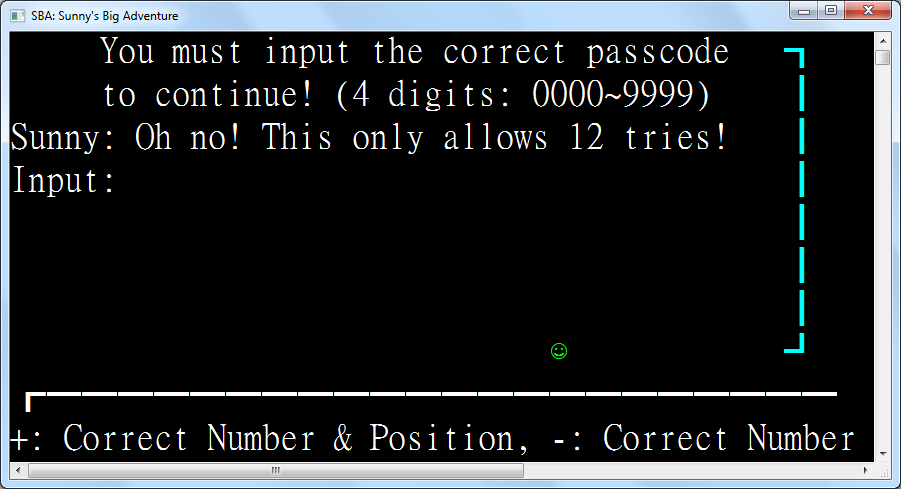
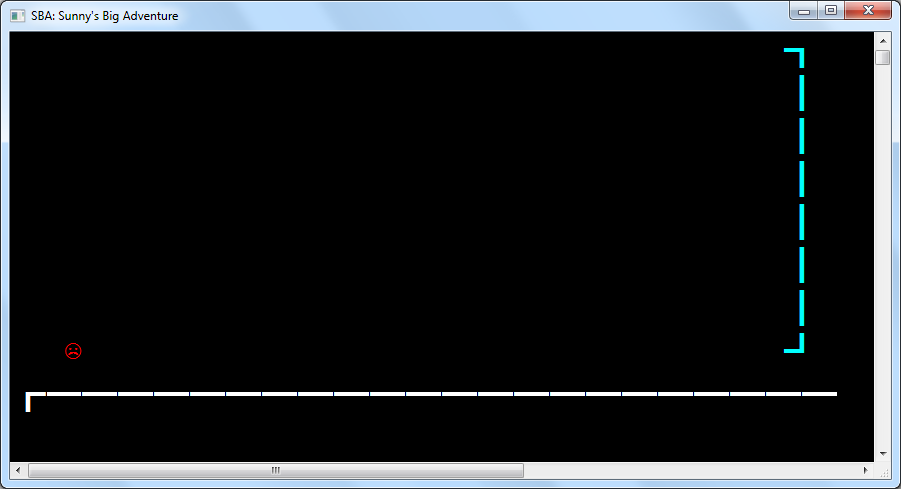
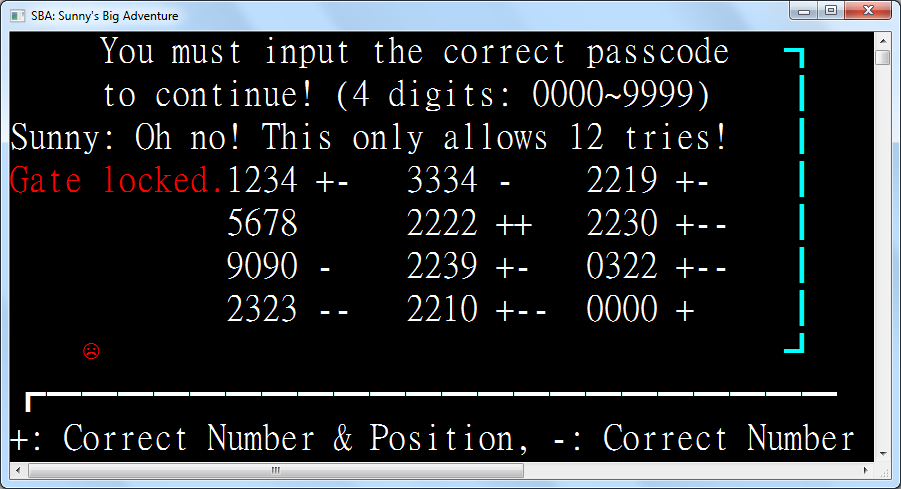
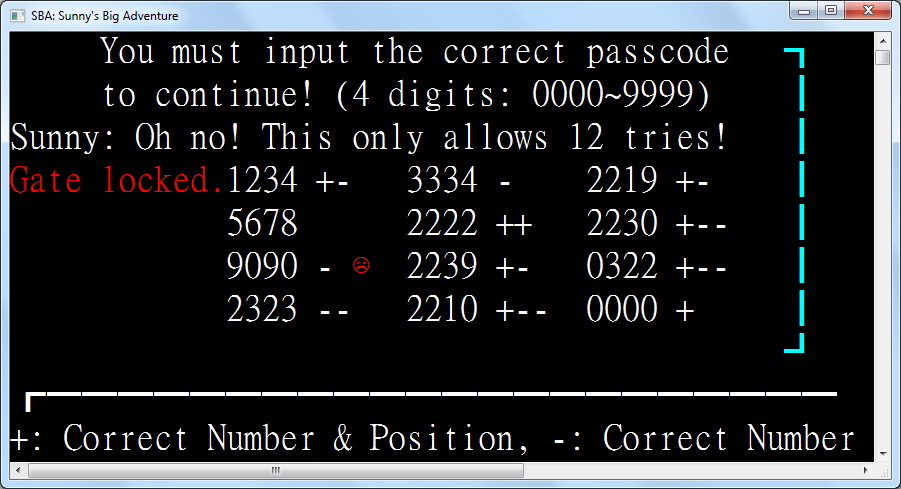
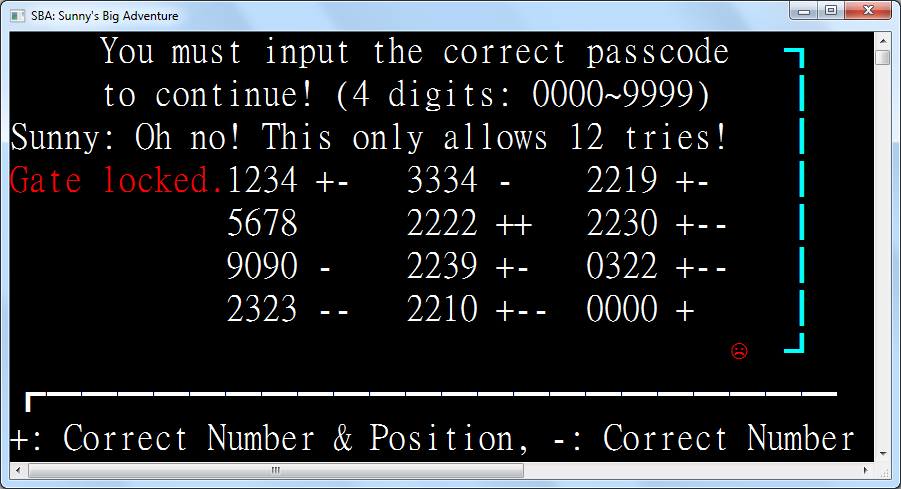
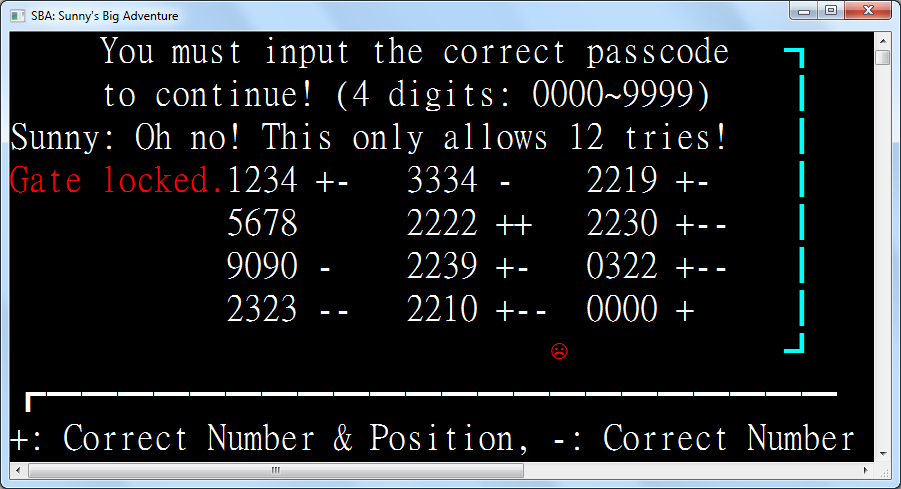
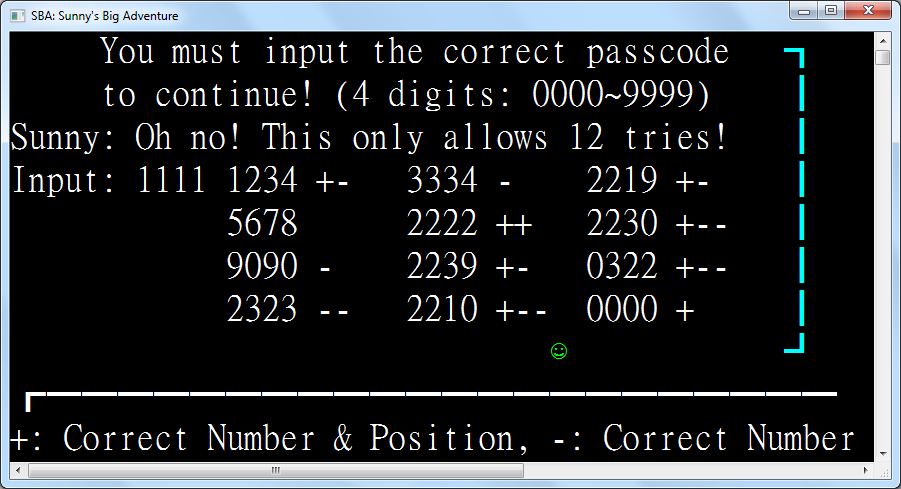
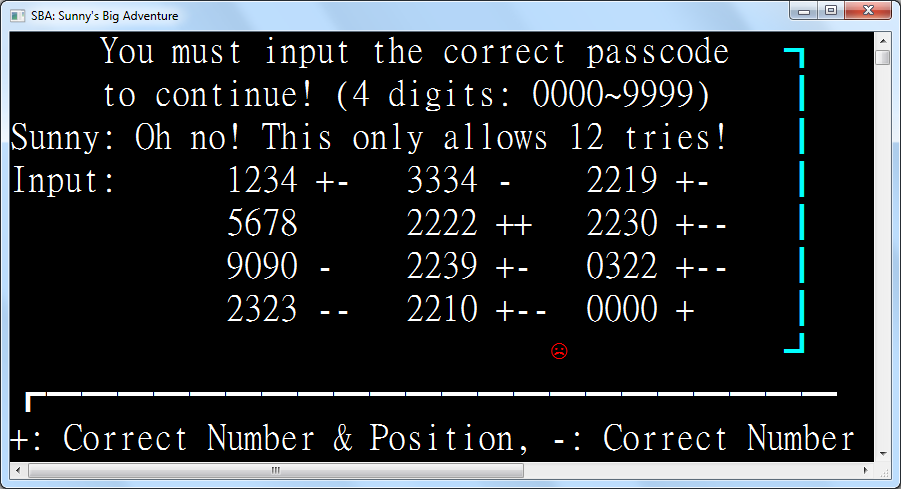
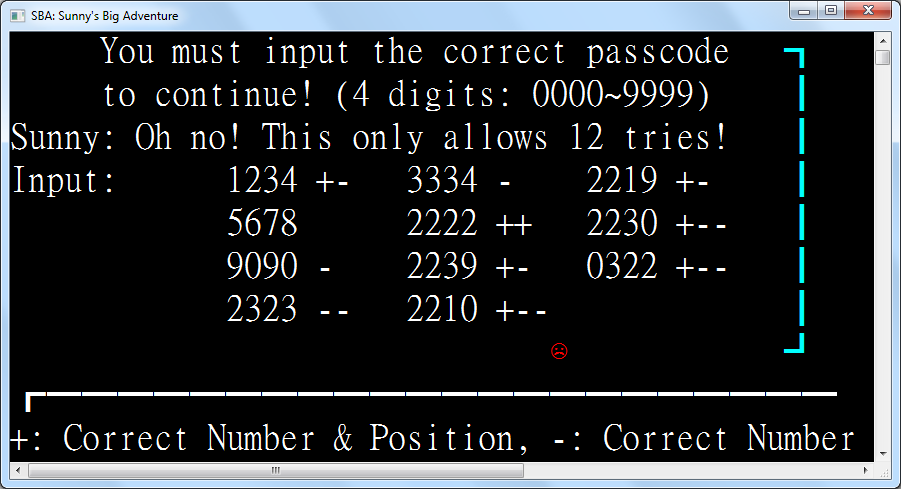
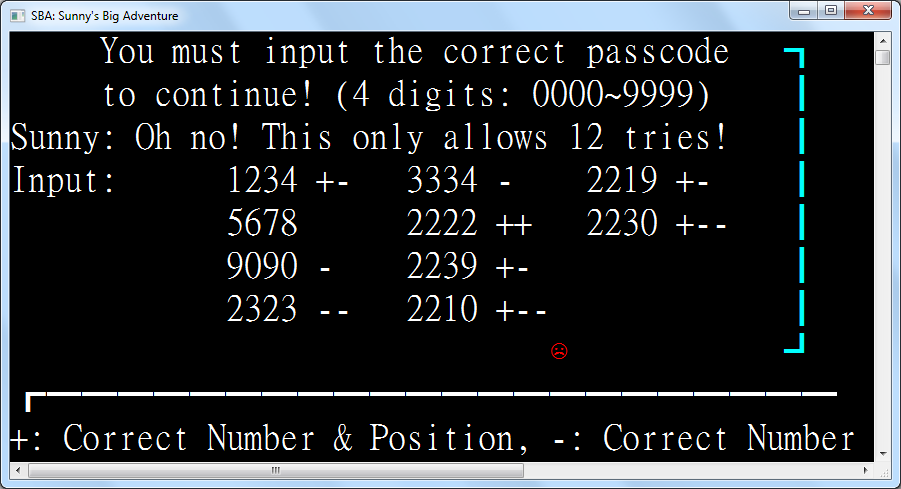
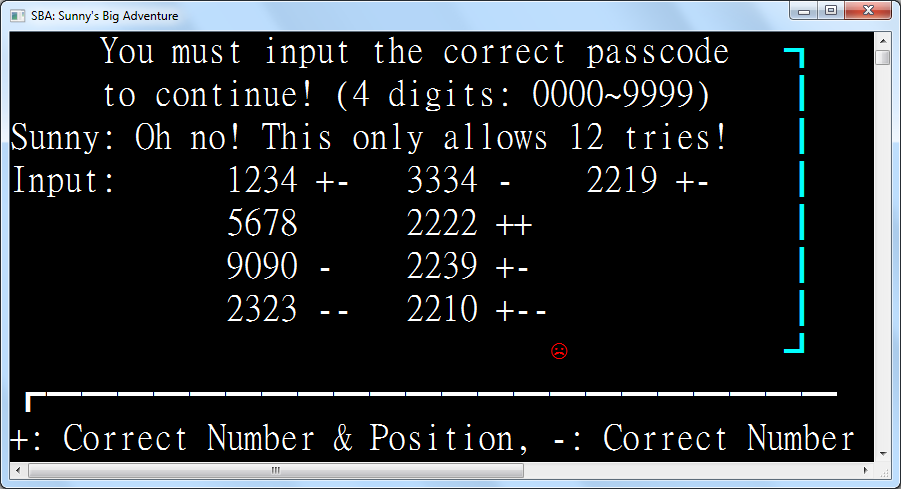
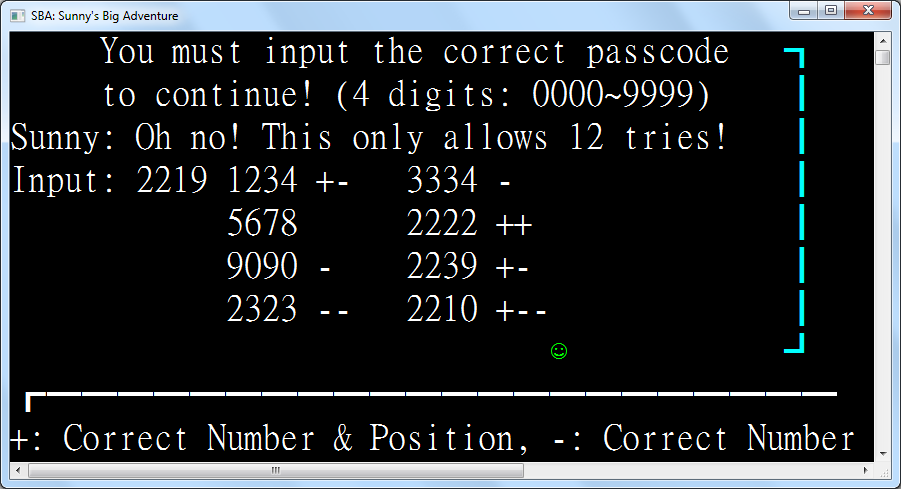
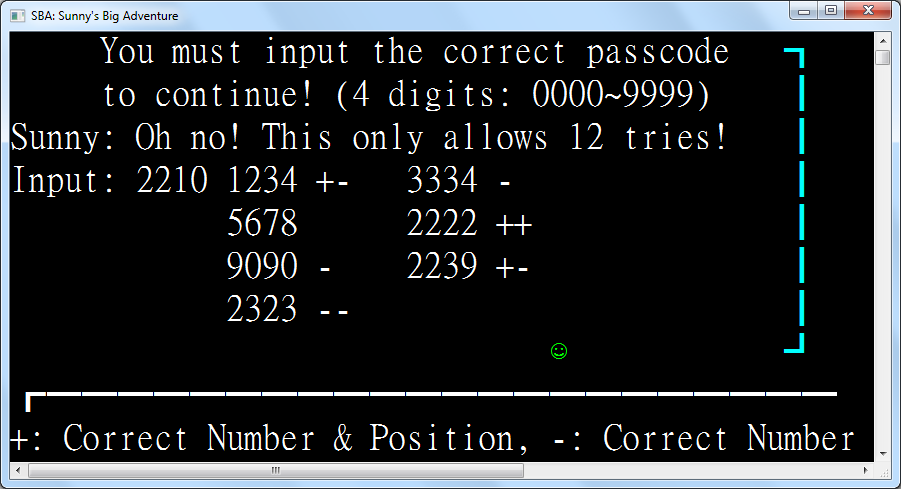
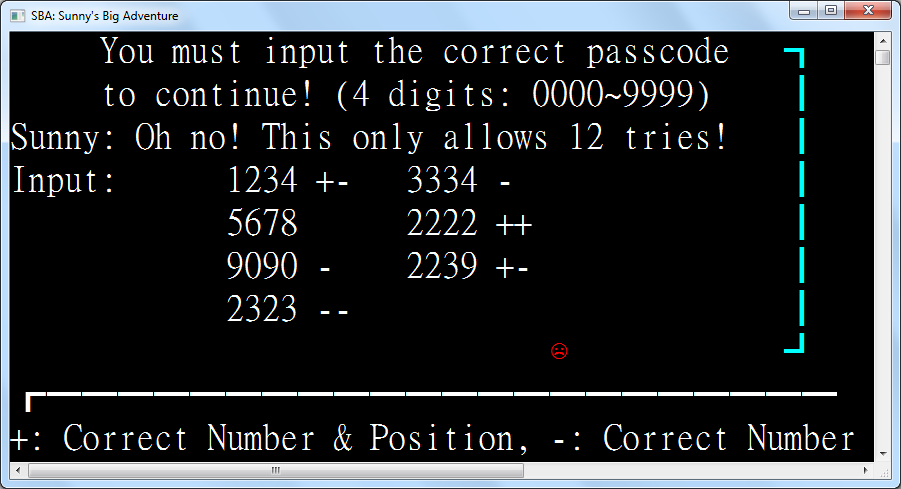
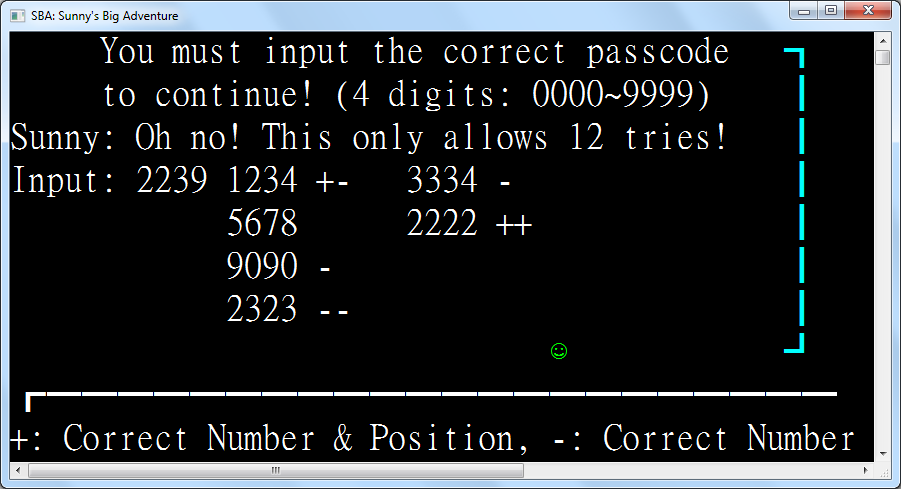
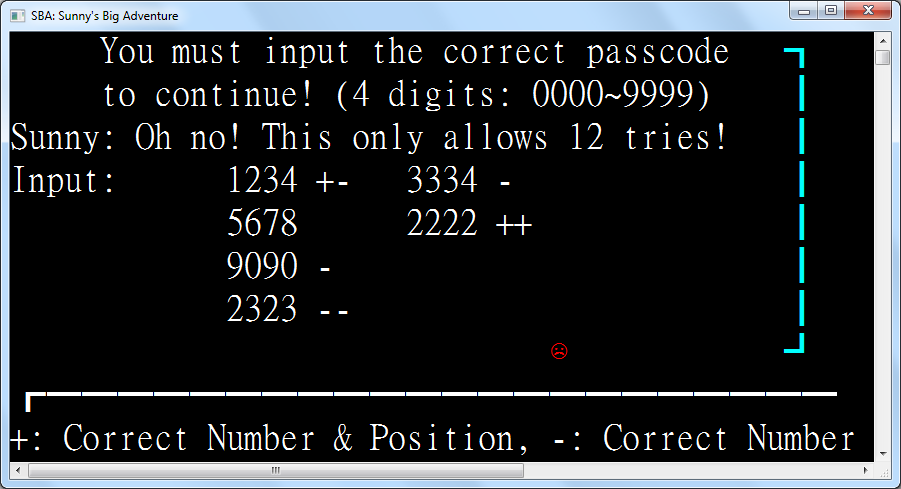
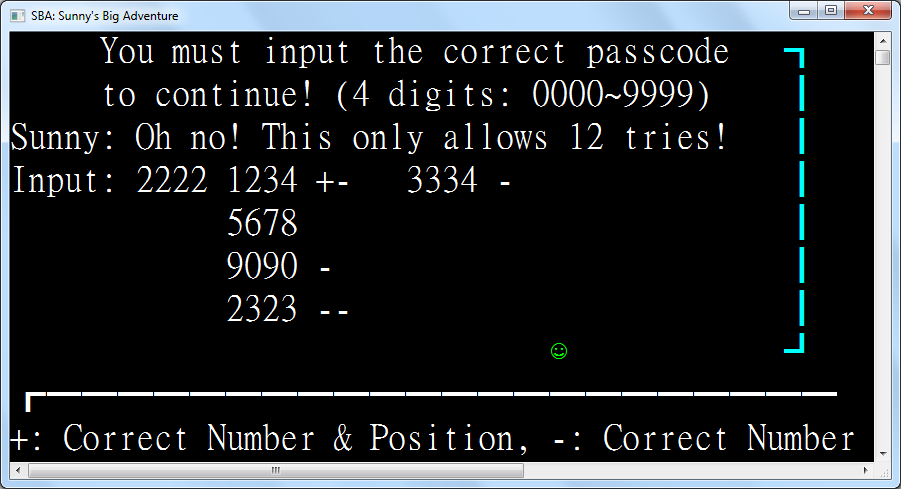
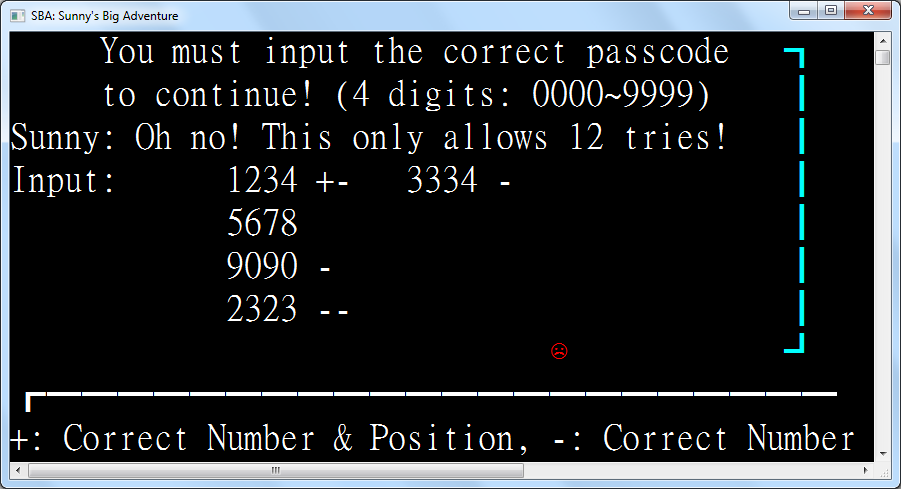
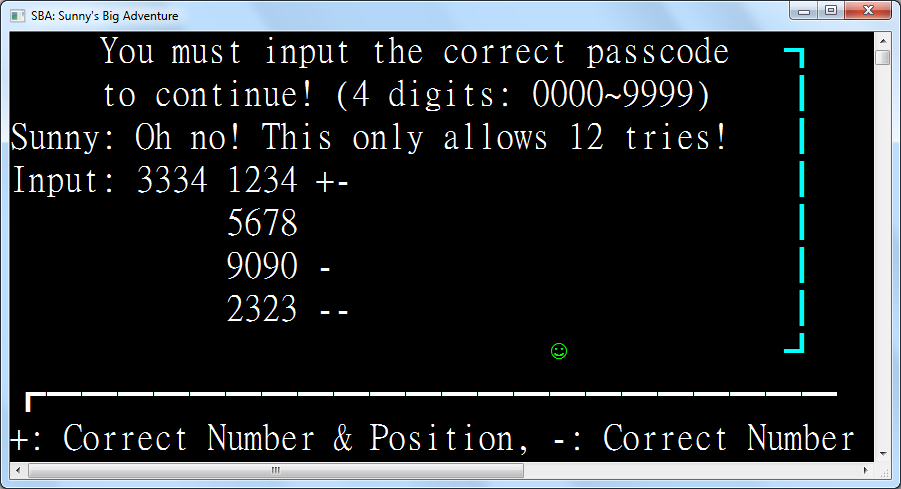
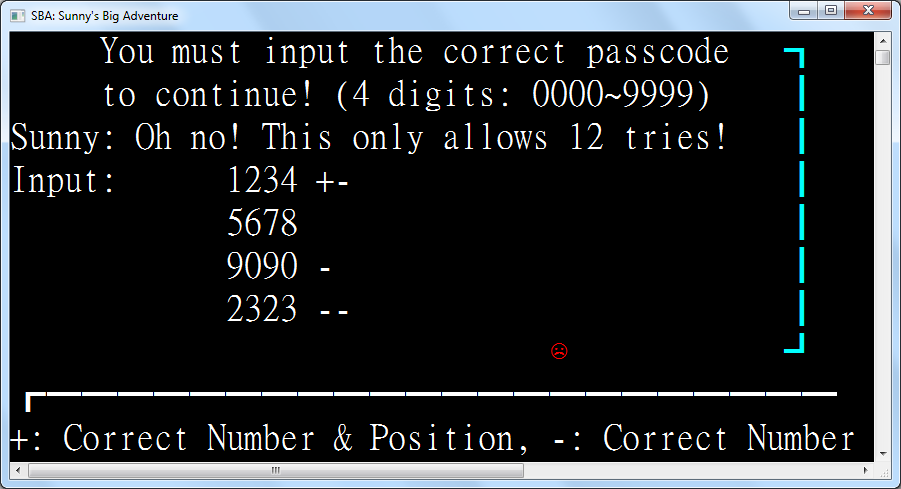
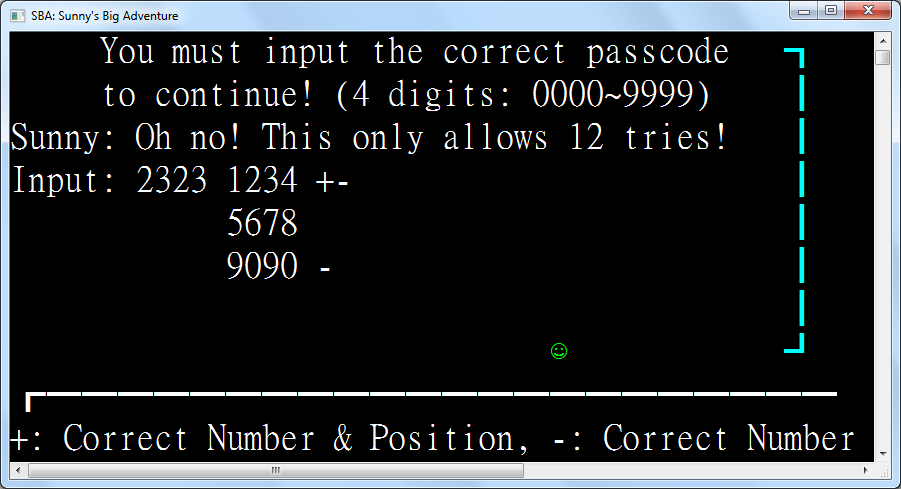
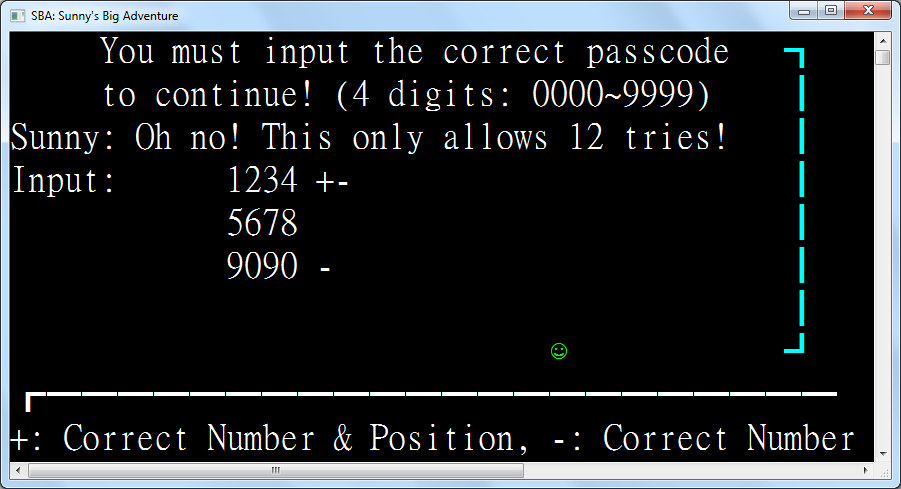
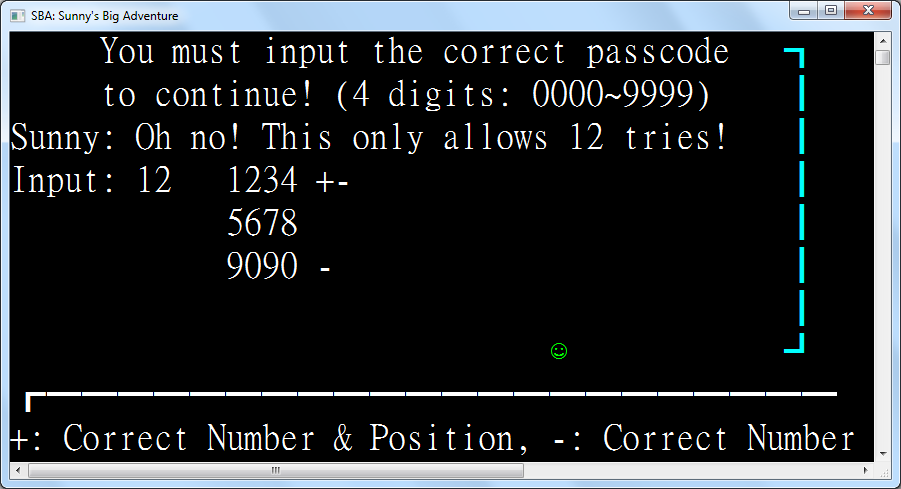
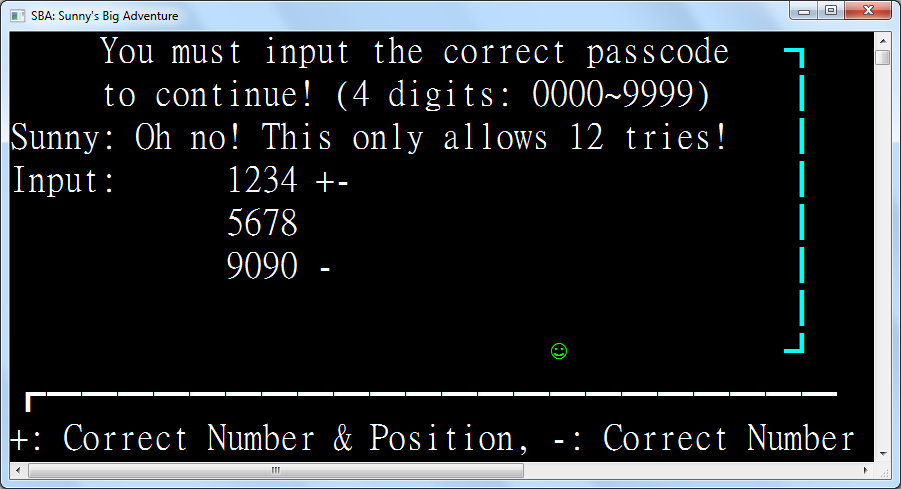
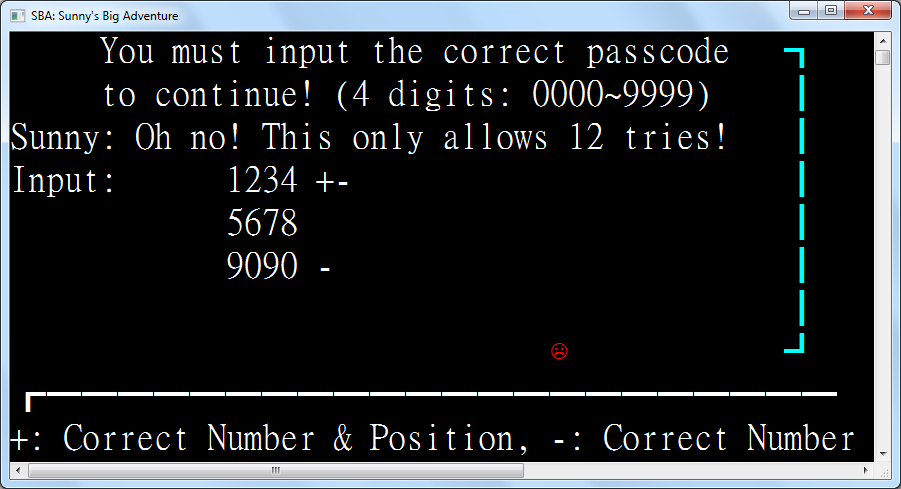
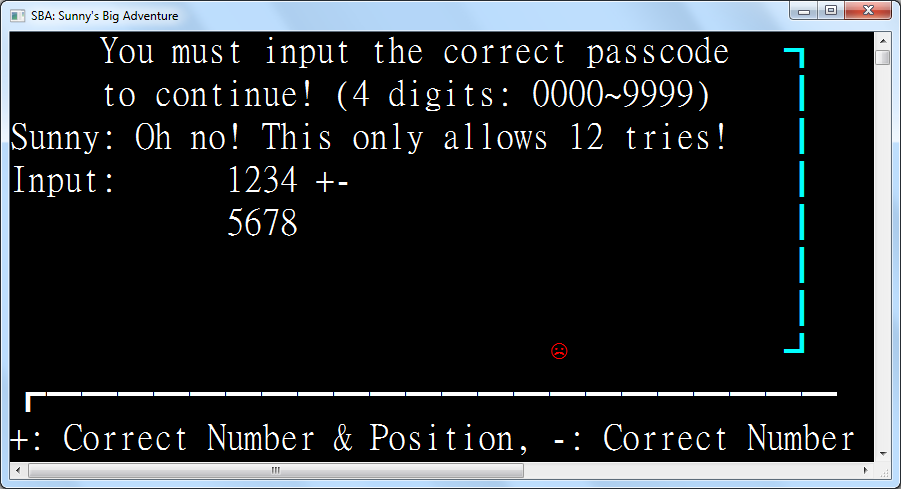
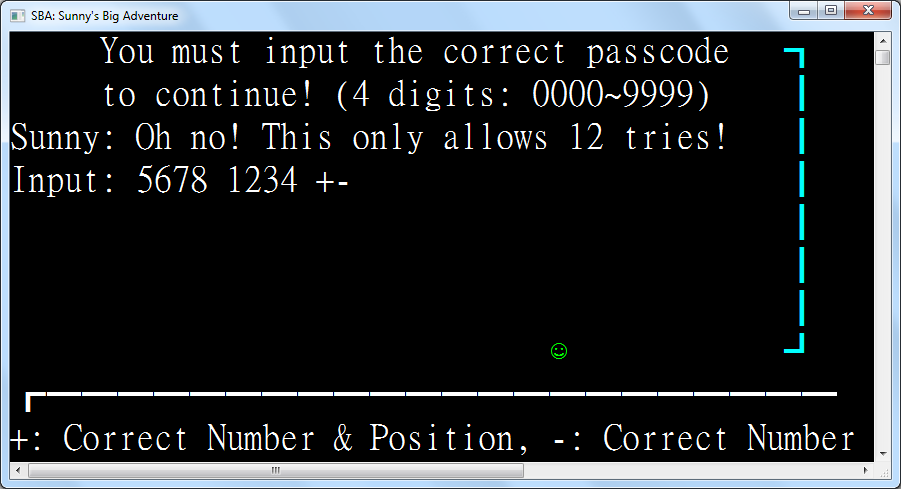
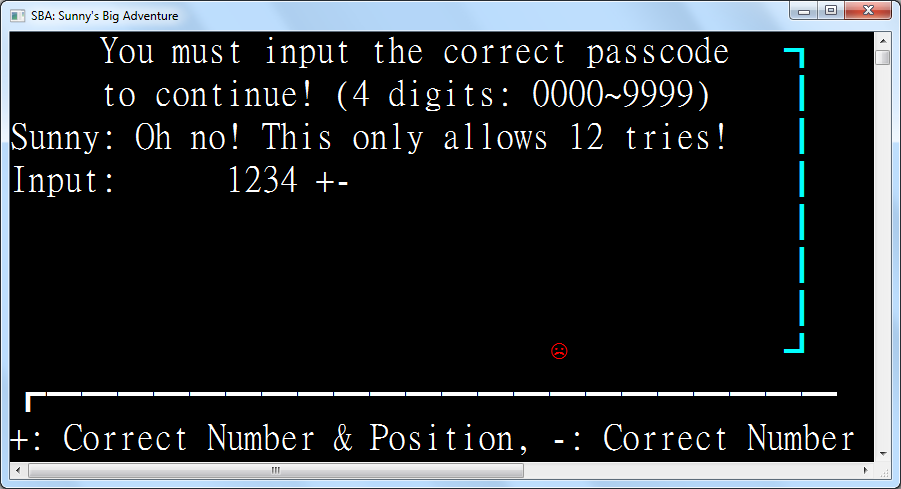
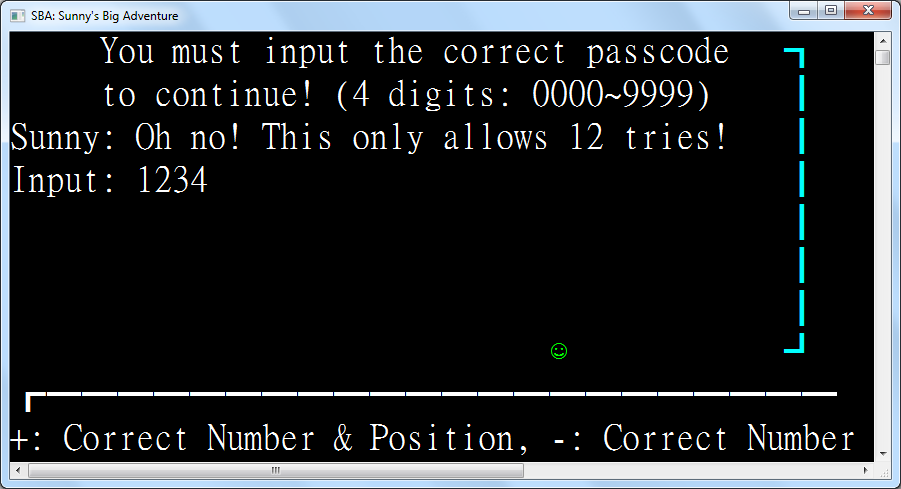
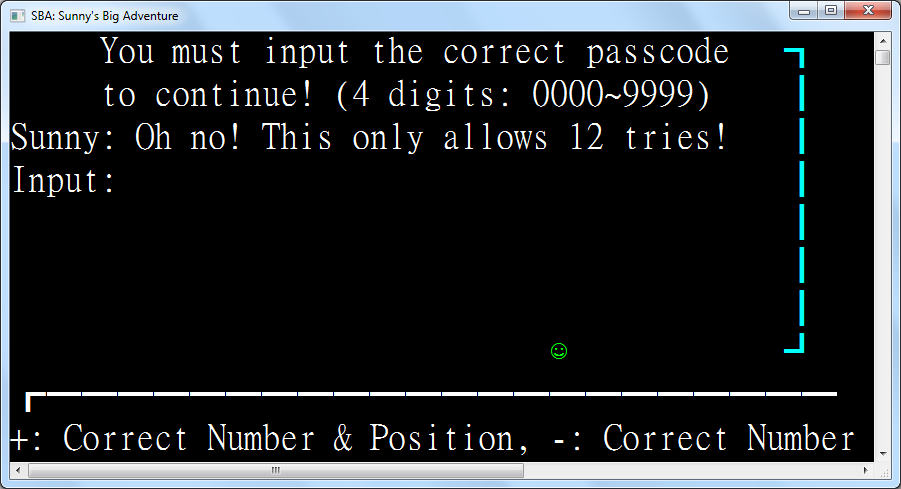
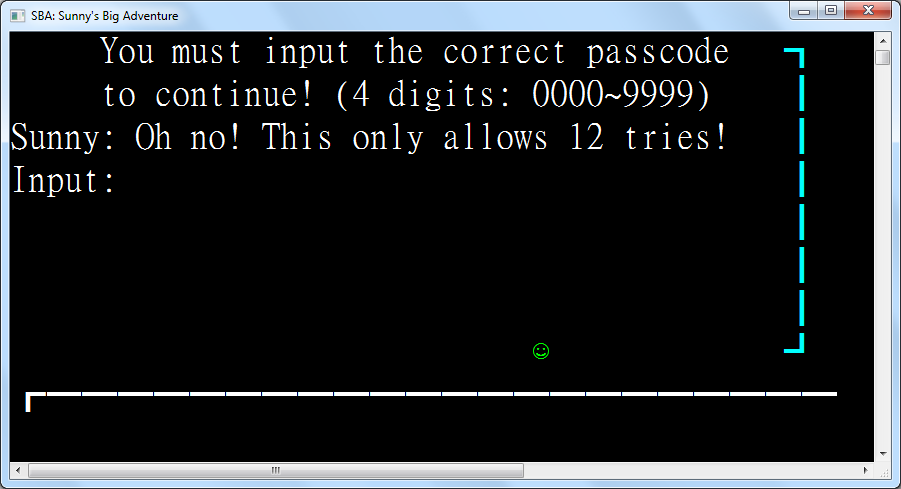
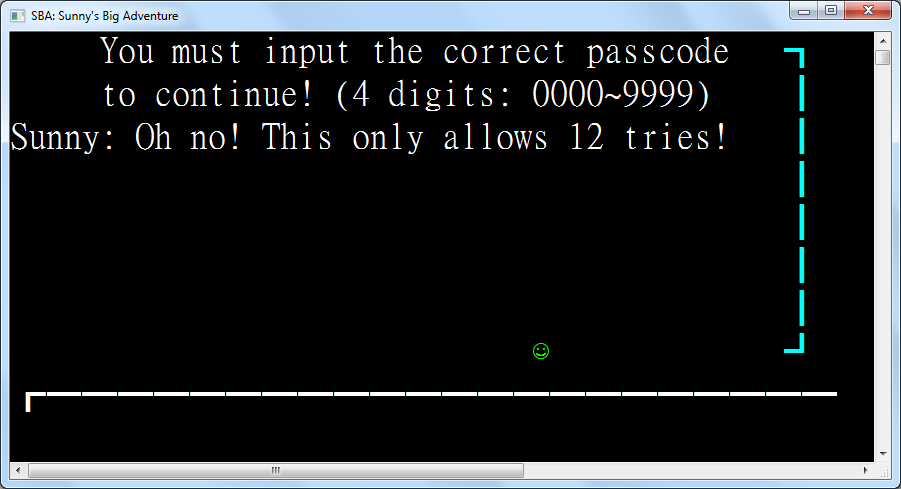
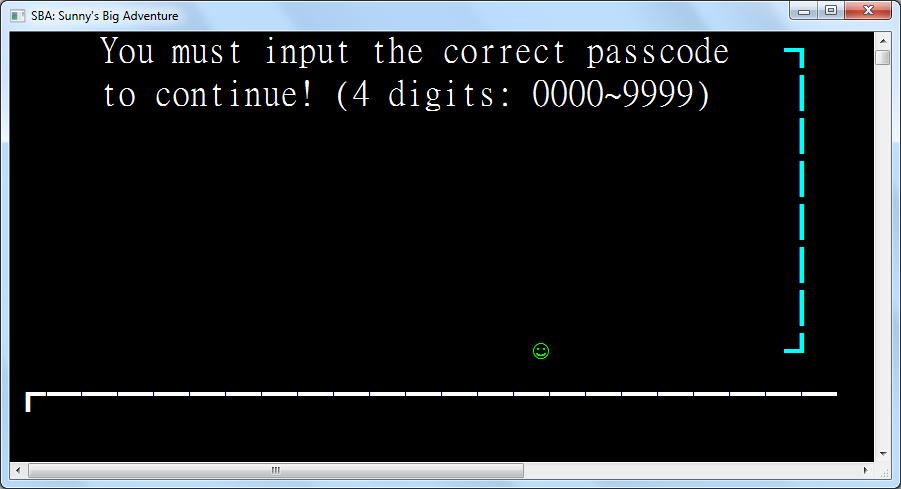
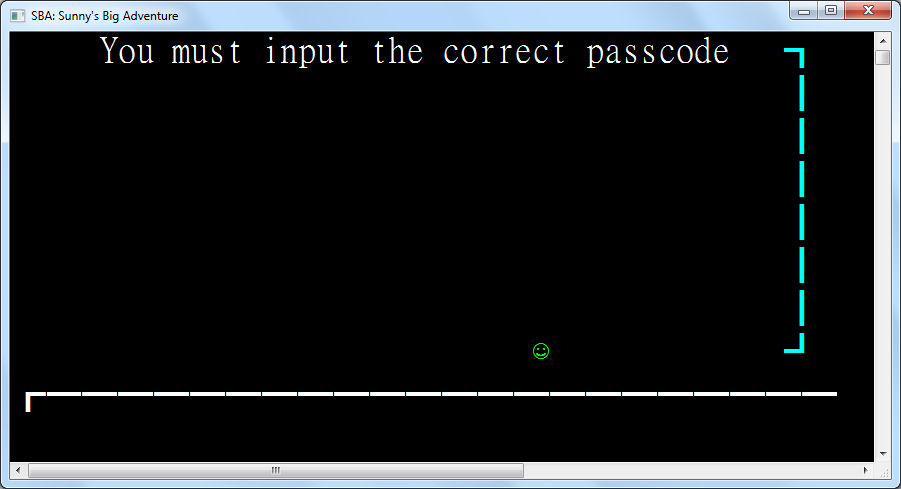
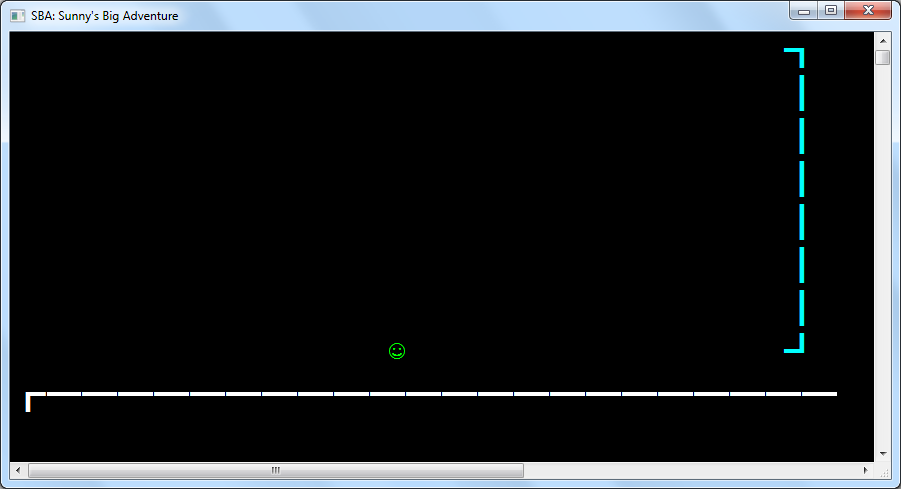
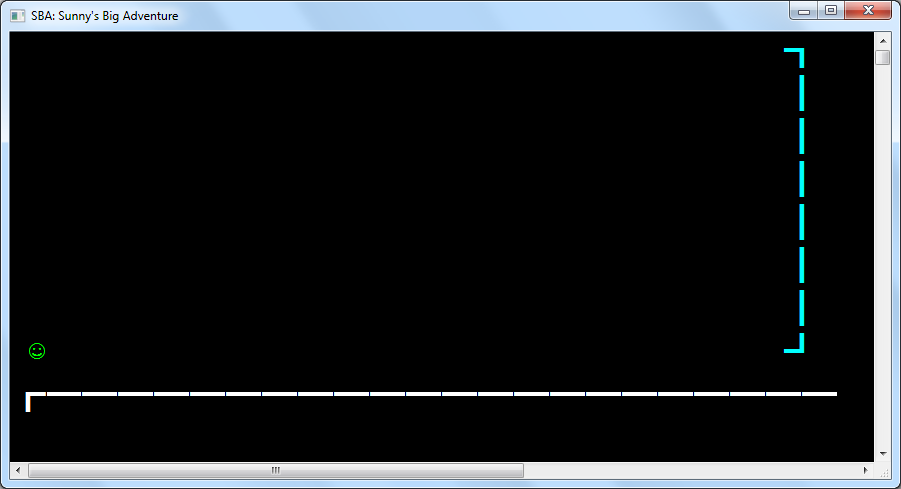
|  |
| --- |
| Dim correctNumCount = g.  GroupBy(Function(gc) gc).  OrderBy(Function(gc) gc.Key).  GroupJoin(a, Function(gc) gc.Key, Function(ac) ac, Function(gc, ac) Math.Min(ac.Count, gc.Count)).  Sum() - correctNumPosCount |

The number of digits with correct value can be obtained by counting the number of each digit in both numbers and calculating the minimum digit frequency for all 10 digits between the two numbers. This number then subtracts the number of digits with correct value and position to avoid repeated counting of digits.

|  |
| --- |
| Return (correctNumPosCount, correctNumCount)  End Function  Protected NextPositionStore As New Point(12, 2)  Function NextPosition() As Point?  Dim p As New Point(NextPositionStore.Left, NextPositionStore.Top + 1)  If p.Top = 7 Then \_  If p.Left = 12 + 2 \* 10 Then NextPosition = Nothing \_  Else NextPosition = New Point(p.Left + 10, 3) Else NextPosition = p  If NextPosition.HasValue Then NextPositionStore = NextPosition.GetValueOrDefault()  End Function |

The NextPosition function generates the next position to display the try. If there is no space left, the gate is locked and the player will have to retry the game with a different number.

|  |
| --- |
| Protected Overrides ReadOnly Property Left As Func(Of Region) = Function() New Region2\_NumberGuess()  Protected Overrides ReadOnly Property Right As Func(Of Region) = Function() New Region4\_ConnectFour()  End Class |



**3.8.5: Region 4 – Connect Four**



This game is controlled by pressing numeric keys to input pieces in a 7 by 7 area. The pieces are placed at the top and fall down until they hit another entity, be it another piece or the border. The green hollow ones are placed by the player while the CPU places the red solid ones. When either participant connects four pieces horizontally, vertically or diagonally, that participant wins. If the CPU wins, the player will need to re-enter the room to retry; if the player wins, Sunny is teleported to the right of the border and can continue on.

|  |
| --- |
| Class Region4\_ConnectFour  Inherits Region  Enum Player  None  Player  CPU  End Enum |

This is the enumeration for calculating the player who won the game, if any.

|  |
| --- |
| Protected ReadOnly GameArea As New Rectangle(8, 1, 17, 8) |

This is the game area. It is referenced by both the border and the trigger zone initiating this game, so to achieve code reuse this is separately defined so any changes can be quickly done.

|  |
| --- |
| Protected ReadOnly Instructions As New TextEntity(WriteEntities, "Press 1~7 to add a piece. Connect 4 to win")  Protected ReadOnly Ruler As New TextEntity(WriteEntities, "11223344556677") |

The ruler provides a convenient way to interpret the number keys so that the player can pinpoint where the piece will be placed. This increases efficiency of playing and avoids counting by hard which bores the player.

|  |
| --- |
| Protected ReadOnly Whites As New GravityEntityFactory(WriteEntities, New Sprite("○"c, ConsoleColor.Green)) |

Internally, the player is called “white” as a reference to the starting player in chess – the white side.

|  |
| --- |
| Protected ReadOnly Blacks As New GravityEntityFactory(WriteEntities, New Sprite("●"c, ConsoleColor.Red)) |

Internally, the CPU is called “black” as a reference to the second player in chess – the black side.

|  |
| --- |
| Protected ReadOnly GameField As New RectangleEntity(WriteEntities, GameArea, ConsoleColor.Blue)  Protected WaitingForCPU As Boolean = False |

The WaitingForCPU flag indicates whether the CPU piece is falling. This avoids the player spamming piece placements to forcibly win the game by placing 4 pieces consecutively before the CPU can react.

|  |
| --- |
| Protected ReadOnly Trigger As New TriggerZone(WriteEntities,  New Rectangle(GameArea.Left - 3, GameArea.Top - 1, GameArea.Width + 6, GameArea.Height + 1),  Function(key)  ActiveEntity.Sprite = Sunny\_ |

As Sunny is sad and angry when failing the game, it should reset its sprite to happy when the game is restarted to mimic human behavior when playing games.

|  |
| --- |
| If Not WaitingForCPU Then  Select Case key  Case ConsoleKey.D1 To ConsoleKey.D7  Dim i = key - ConsoleKey.D0  If Not Entities.Any(Function(e) If(e.Position = New Point(GameArea.Left + i \* 2, GameArea.Top + 1), False)) Then  Whites.Add(New Point(GameArea.Left + i \* 2, GameArea.Top + 1), AddressOf OnCPUTick)  WaitingForCPU = True  End If  End Select  End If |

When a numeric key between 1 and 7 is pressed, a piece in the corresponding column will be placed if there is no other piece obstructing piece placement. The WaitingForCPU flag will be set to prohibit key spamming.

|  |
| --- |
| Return True  End Function,  Sub()  Instructions.Position = New Point(3, 0)  Ruler.Position = New Point(10, 9)  AddHandler Tick, AddressOf OnTick |

When Sunny approaches the game border, the instructions will appear and the game will start checking for whether a participant is winning by adding OnTick onto the Tick event.

|  |
| --- |
| End Sub, Sub() RemoveHandler Tick, AddressOf OnTick) |

When Sunny wins and exits the game, the game will stop checking for whether a participant is winning by removing OnTick from the Tick event to avoid a piece placed by the CPU after the player winning interfering with the game results.

|  |
| --- |
| Protected Overrides ReadOnly Property Left As Func(Of Region) = Function() New Region3\_BullsAndCows()  Protected Overrides ReadOnly Property Right As Func(Of Region) = Function() New Region5\_Win()  Protected Sub OnTick()  Select Case WhoWin(Nothing)  Case Player.Player  Instructions.Color = ConsoleColor.Green  Instructions.Text = "You win!"  Instructions.Position = New Point(13, 0)  ActiveEntity.Position = New Point(GameArea.Right + 4, GameArea.Bottom - 3)  Trigger.Dispose() |

When the player wins, "You win!" in green will be displayed on top of the game area to congratulate the player and Sunny will be moved to the right of the game area.

|  |
| --- |
| Case Player.CPU  Instructions.Color = ConsoleColor.Red  Instructions.Text = "CPU wins!"  Instructions.Position = New Point(13, 0)  ActiveEntity.Sprite = Sunny\_Angry  Trigger.Dispose() |

When the CPU wins, "CPU wins!" in red will be displayed on top of the game area to shame the player and Sunny will be sad and angry. Sunny will need to reenter the room to start the game again.

|  |
| --- |
| End Select  End Sub  Protected Function WhoWin(simulateCoordinatesOpt As (blackX As Integer?, whiteX As Integer?)) As Player |

The WhoWin function returns the player who won the game, if any, and optionally takes a possible black move followed by a white move to aid the CPU in simulating the possible consequences of a move.

|  |
| --- |
| Dim Matches =  Function(side As GravityEntityFactory, point As Point, simulatePosition As Point?)  Dim occupier As GravityEntityFactory.GravityEntityFactoryEntity = Nothing  For Each item In Entities.OfType(Of GravityEntityFactory.GravityEntityFactoryEntity)  If item.Position = New Point(GameArea.Left + point.Left \* 2, GameArea.Top - 1 + point.Top) Then  occupier = item  Exit For  End If  Next |

To detect if a piece from a provided side is located in a specified coordinate, this function runs a linear search to check the coordinates of each entity.

|  |
| --- |
| Return If(occupier IsNot Nothing,  occupier.Owner Is side AndAlso occupier.VerticalVelocity = 0,  point = simulatePosition) |

To avoid registering falling pieces as a valid piece, the piece is checked for whether it is falling.

|  |
| --- |
| End Function  Dim black = IfHasValue(simulateCoordinatesOpt.blackX,  Function(blackX)  For y = 1 To 7  If Matches(Whites, New Point(blackX, y), Nothing) OrElse  Matches(Blacks, New Point(blackX, y), Nothing) Then  Return New Point(blackX, If(y = 1, 1, y - 1))  End If  Next  Return New Point(blackX, 7)  End Function) |

black is the resulting coordinates of the black y-coordinate provided, with all other pieces taken into account.

|  |
| --- |
| Dim white = IfHasValue(simulateCoordinatesOpt.whiteX,  Function(whiteX)  For y = 1 To 7  If Matches(Whites, New Point(whiteX, y), Nothing) OrElse  Matches(Blacks, New Point(whiteX, y), Nothing) OrElse  black = New Point(whiteX, y) Then \_  Return New Point(whiteX, If(y = 1, 1, y - 1))  Next  Return New Point(whiteX, 7)  End Function) |

white is the resulting coordinates of the white y-coordinate provided, with all other pieces taken into account.

|  |
| --- |
| Dim connected = Function(p1 As Point, p2 As Point, p3 As Point, p4 As Point)  If Matches(Whites, p1, white) AndAlso  Matches(Whites, p2, white) AndAlso  Matches(Whites, p3, white) AndAlso  Matches(Whites, p4, white) Then Return Player.Player  If Matches(Blacks, p1, black) AndAlso  Matches(Blacks, p2, black) AndAlso  Matches(Blacks, p3, black) AndAlso  Matches(Blacks, p4, black) Then Return Player.CPU  Return Player.None  End Function |

connected is a function for checking if a participant has pieces placed in the four coordinates.

|  |
| --- |
| ' -  For y = 1 To 7  For x = 1 To 4  WhoWin = connected(New Point(x, y), New Point(x + 1, y), New Point(x + 2, y), New Point(x + 3, y))  If WhoWin <> Player.None Then Return WhoWin  Next  Next |

First, all horizontal sequences of 4 pieces are checked with a for-loop for any possible winners.

|  |
| --- |
| ' |  For y = 1 To 4  For x = 1 To 7  WhoWin = connected(New Point(x, y), New Point(x, y + 1), New Point(x, y + 2), New Point(x, y + 3))  If WhoWin <> Player.None Then Return WhoWin  Next  Next |

Then, all vertical sequences of 4 pieces are checked with a for-loop for any possible winners.

|  |
| --- |
| ' \  For x = 1 To 4  For y = 1 To 4  WhoWin = connected(New Point(x, y), New Point(x + 1, y + 1), New Point(x + 2, y + 2), New Point(x + 3, y + 3))  If WhoWin <> Player.None Then Return WhoWin  Next  Next |

Next, all main diagonal sequences of 4 pieces are checked with a for-loop for any possible winners.

|  |
| --- |
| ' /  For x = 1 To 4  For y = 4 To 7  WhoWin = connected(New Point(x, y), New Point(x + 1, y - 1), New Point(x + 2, y - 2), New Point(x + 3, y - 3))  If WhoWin <> Player.None Then Return WhoWin  Next  Next |

After that, all anti-diagonal sequences of 4 pieces are checked with a for-loop for any possible winners.

|  |
| --- |
| Return Player.None |

Finally, when no winner is found, the result indicating that neither participant won is returned.

|  |
| --- |
| End Function  Sub OnCPUTick()  Dim cpu = CPUTurn()  If cpu IsNot Nothing Then  Blacks.Add(New Point(GameArea.Left + cpu.GetValueOrDefault() \* 2, GameArea.Top + 1))  Else  Instructions.Text = "Stalemate."  Instructions.Position = New Point(12, 0)  Trigger.Dispose()  End If  RemoveHandler Tick, AddressOf OnCPUTick  WaitingForCPU = False  End Sub |

When a CPU Tick occurs, the function called CPUTurn is called to obtain the CPU piece placement. If the CPU cannot place a piece, then a stalemate is detected and the user will have to reenter the room to start again.

|  |
| --- |
| Function CPUTurn() As Integer?  Dim choices = Enumerable.Range(1, 7).Where(Function(x) \_  If(Not (Whites.ItemAt(New Point(GameArea.Left + x \* 2, GameArea.Top + 1))?.VerticalVelocity = 0 OrElse  Blacks.ItemAt(New Point(GameArea.Left + x \* 2, GameArea.Top + 1))?.VerticalVelocity = 0), True)).ToHashSet()  Dim avoidChoices = New HashSet(Of Integer)() |

When the CPU chooses a column to place a piece, the available choices are enumerated and filtered based on whether that column is full or not.

|  |
| --- |
| ' 1. Black win  For Each x In choices  If WhoWin((x, Nothing)) = Player.CPU Then Return x  Next |

The CPU will first try to immediately win.

|  |
| --- |
| ' 2. Block white win  For Each x In choices  If WhoWin((Nothing, x)) = Player.Player Then Return x  Next |

If it can’t win immediately, the CPU will then try to block the player from immediately winning.

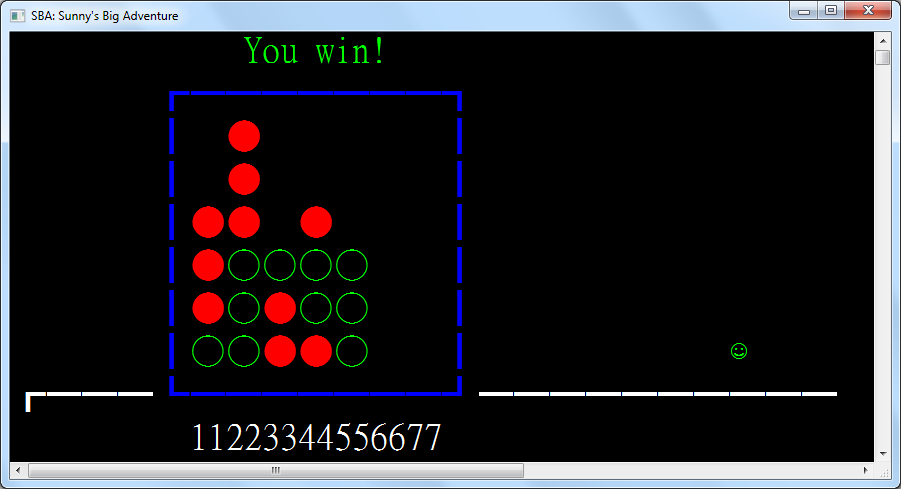
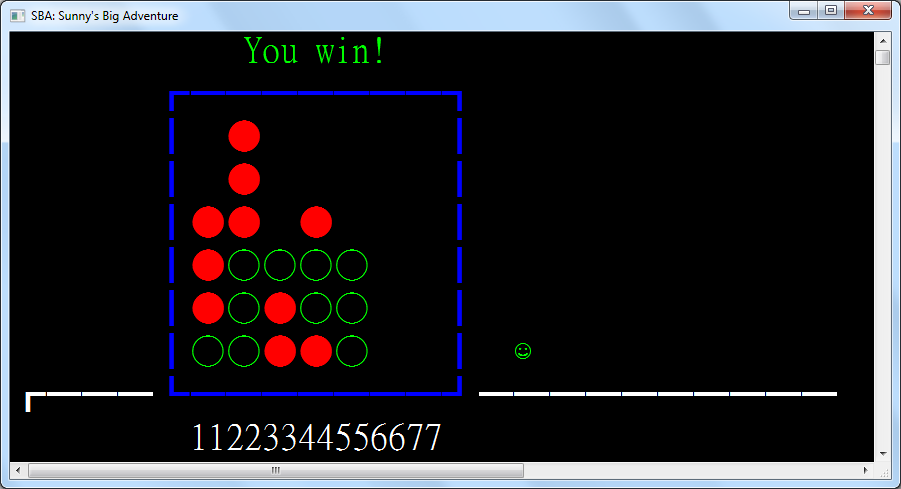
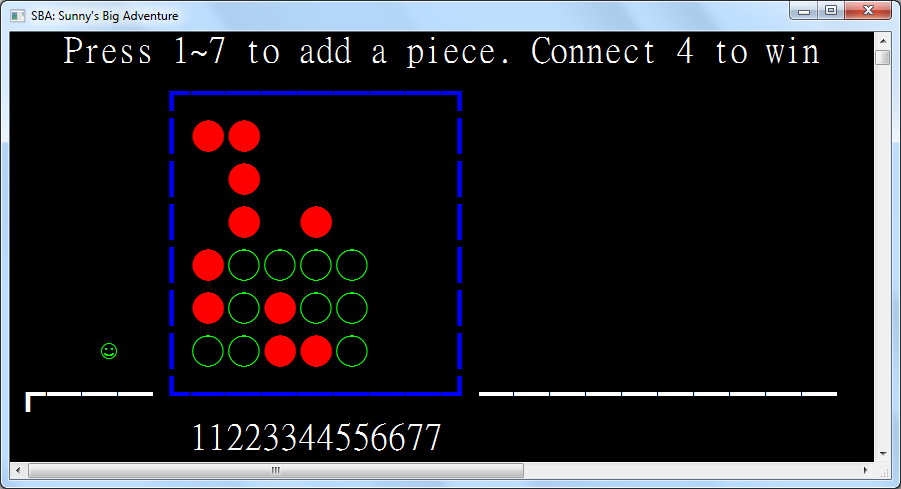
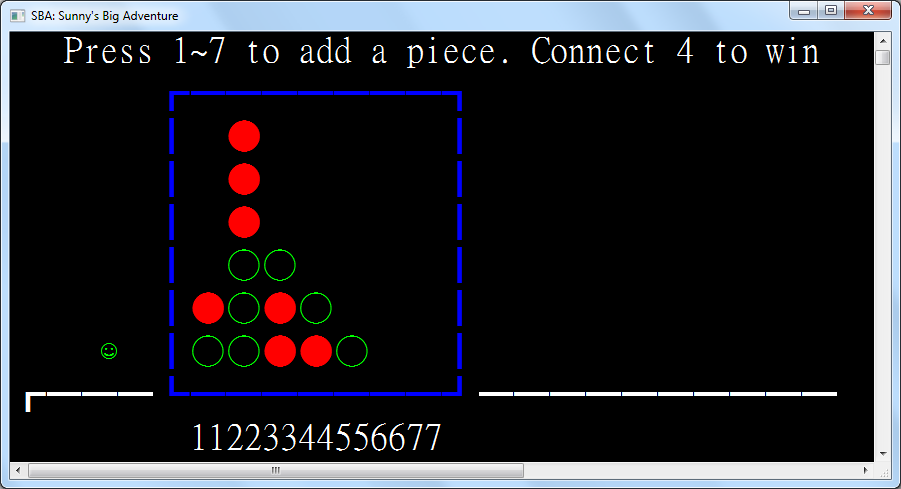
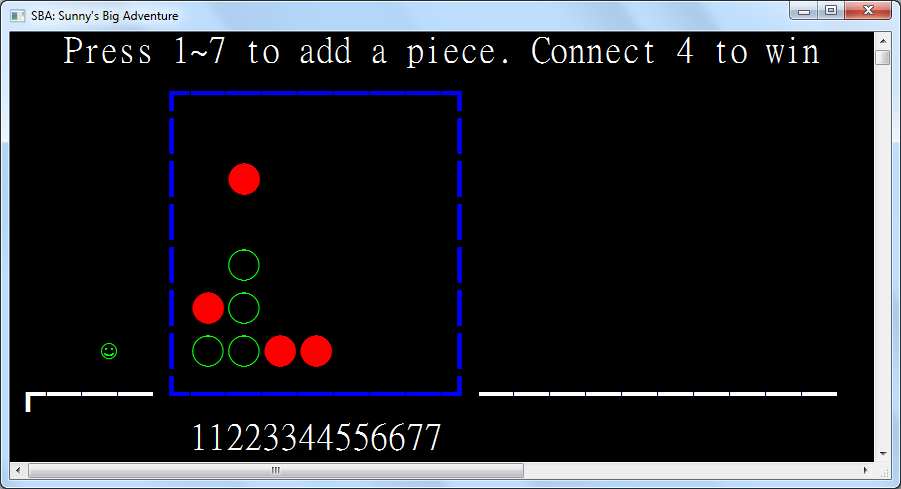
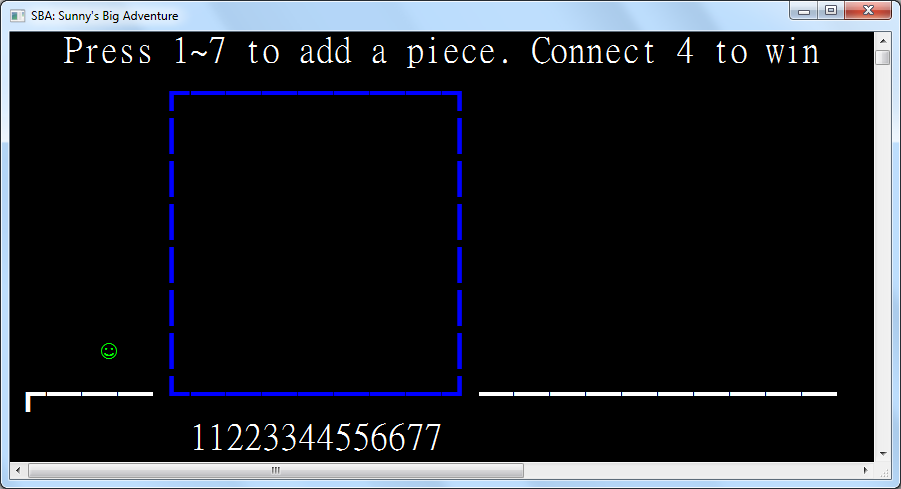
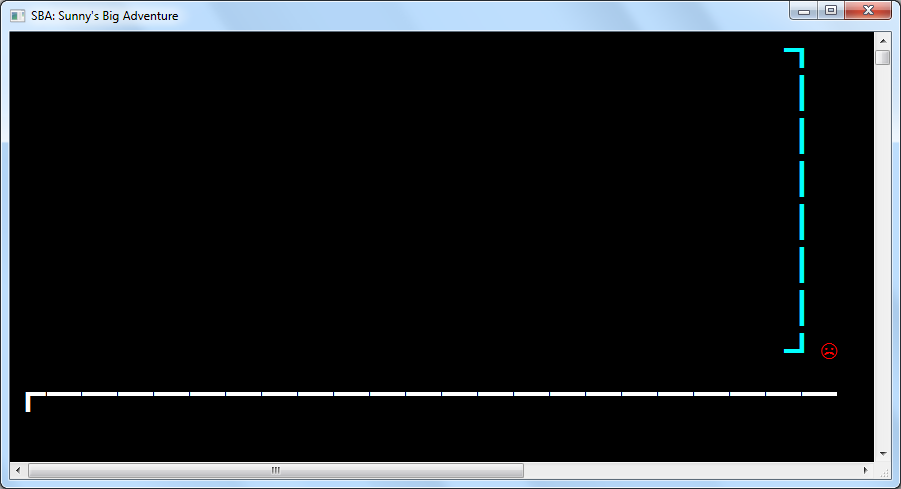
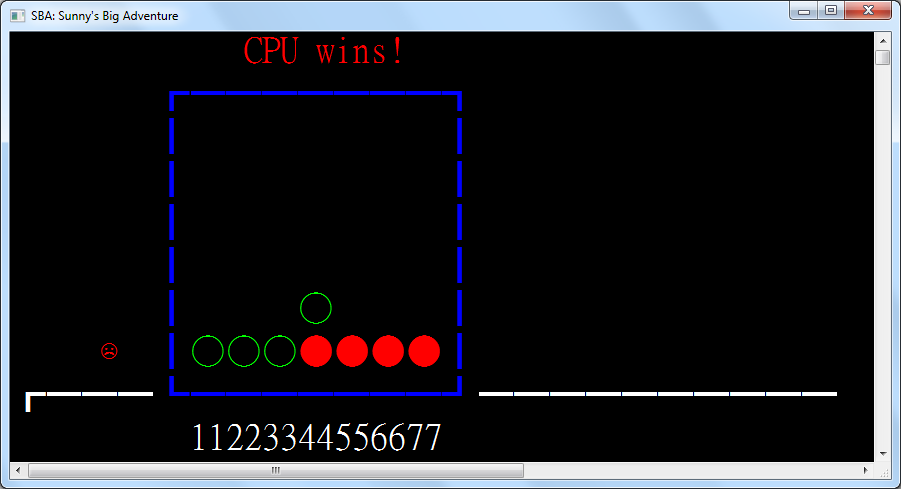
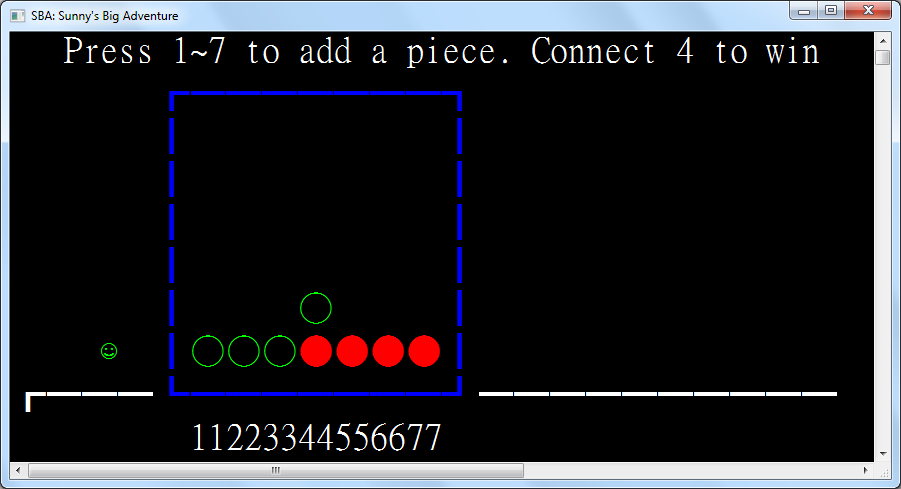
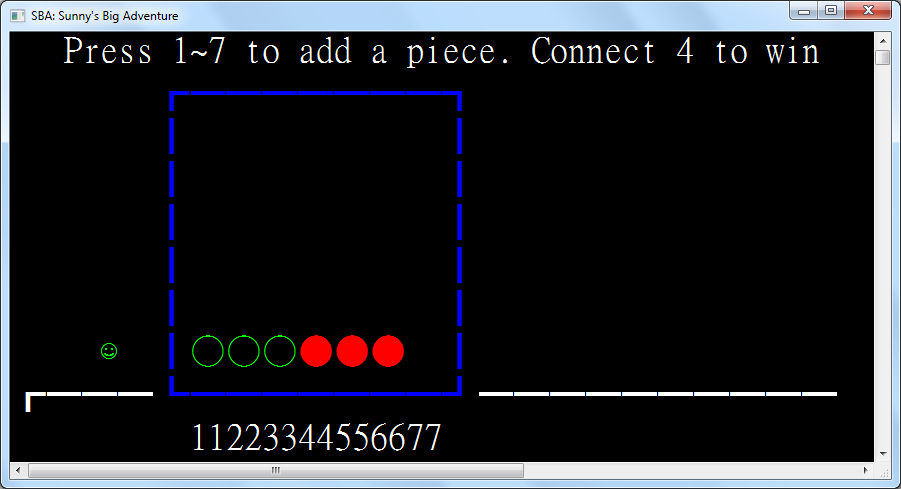
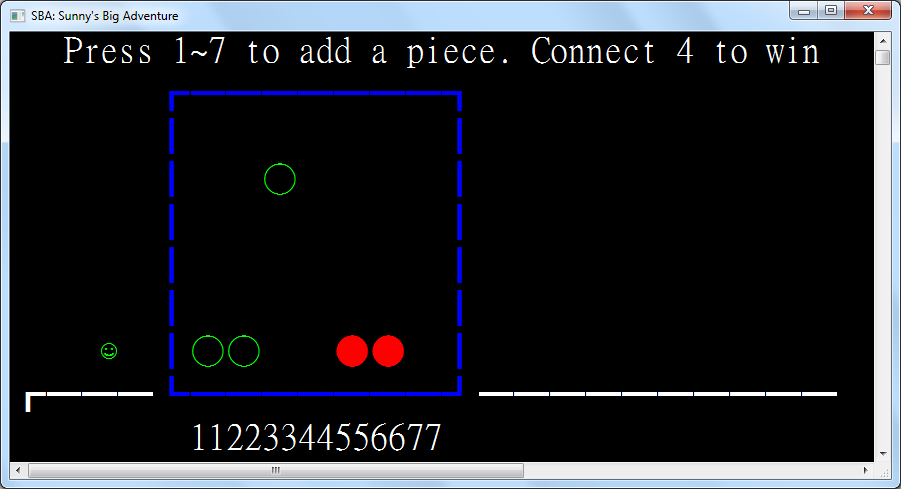
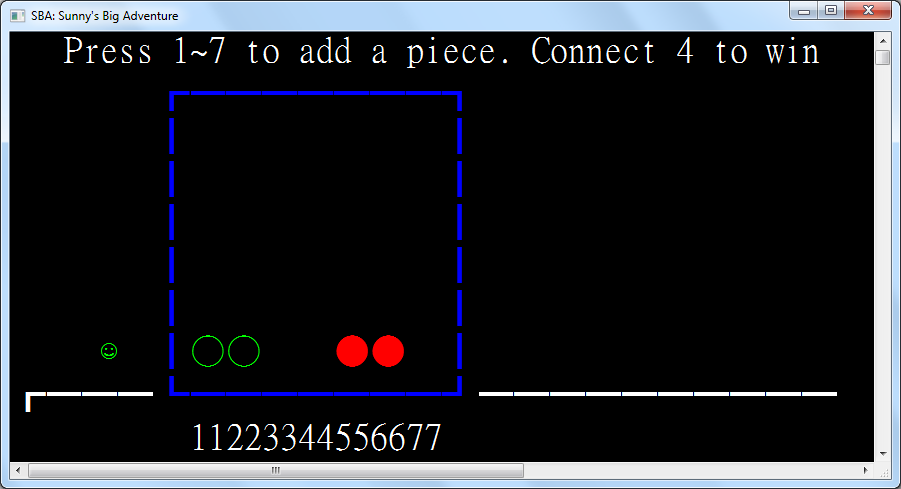
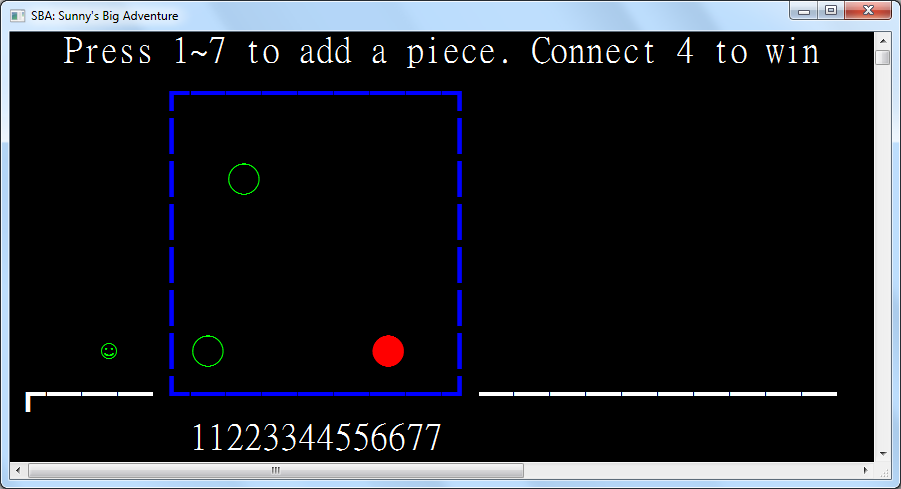
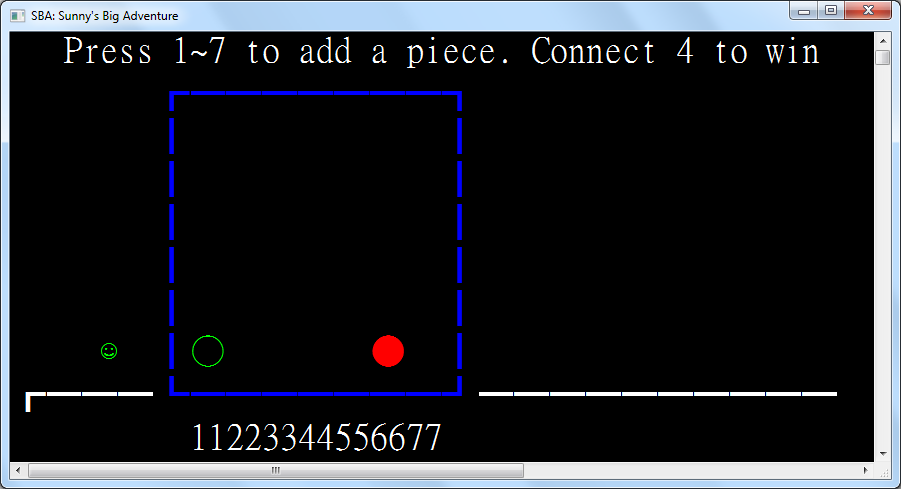
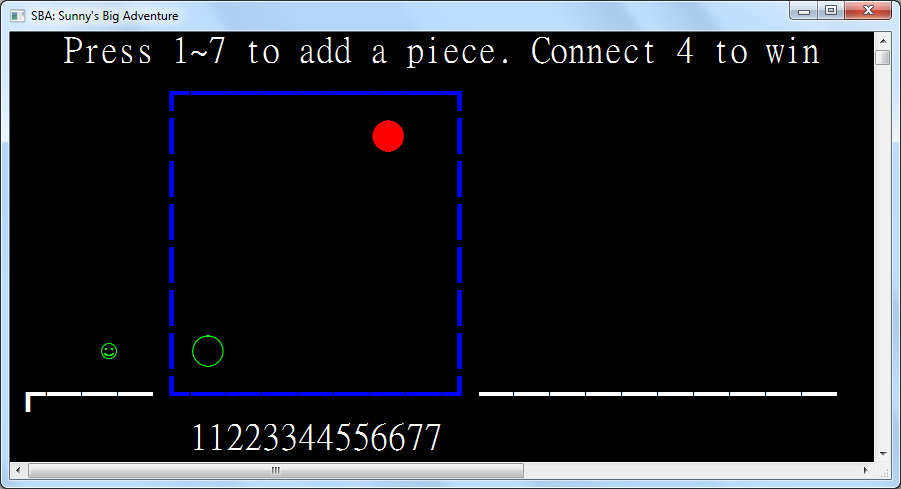
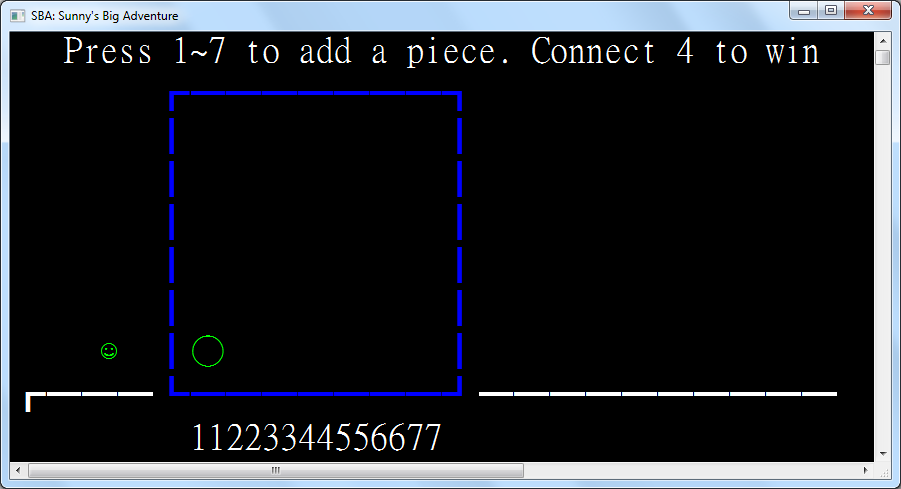
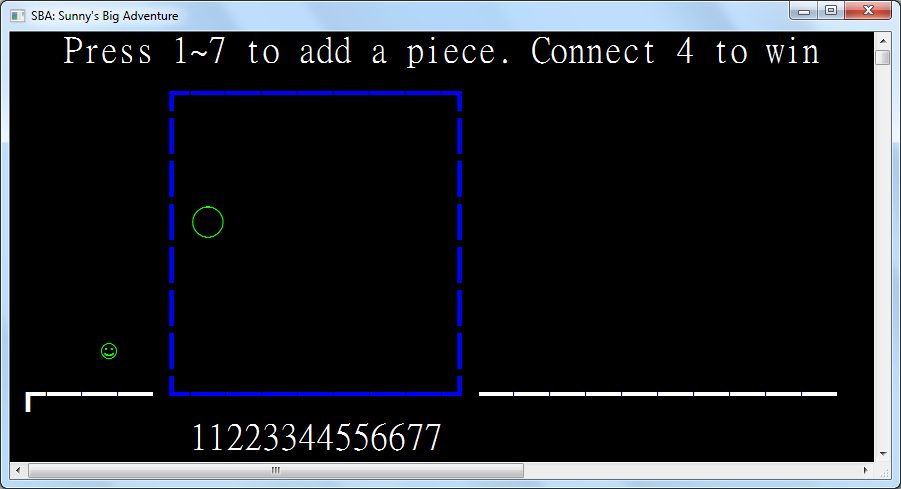
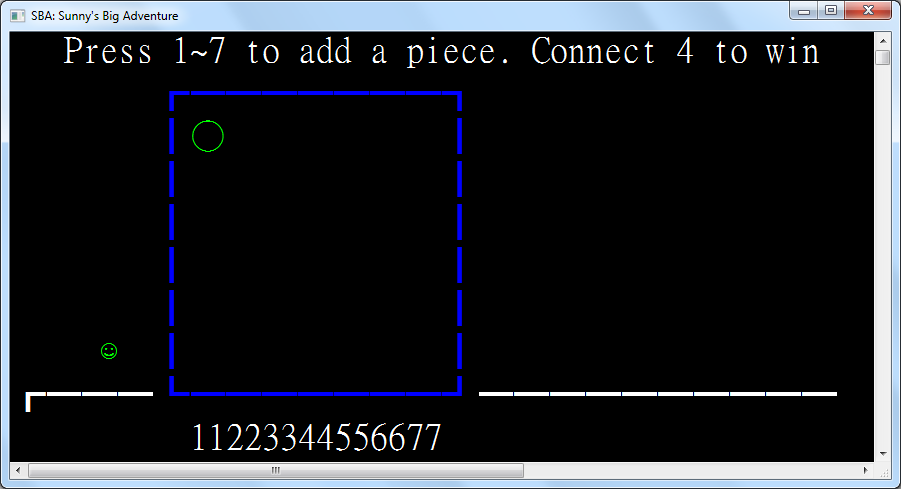
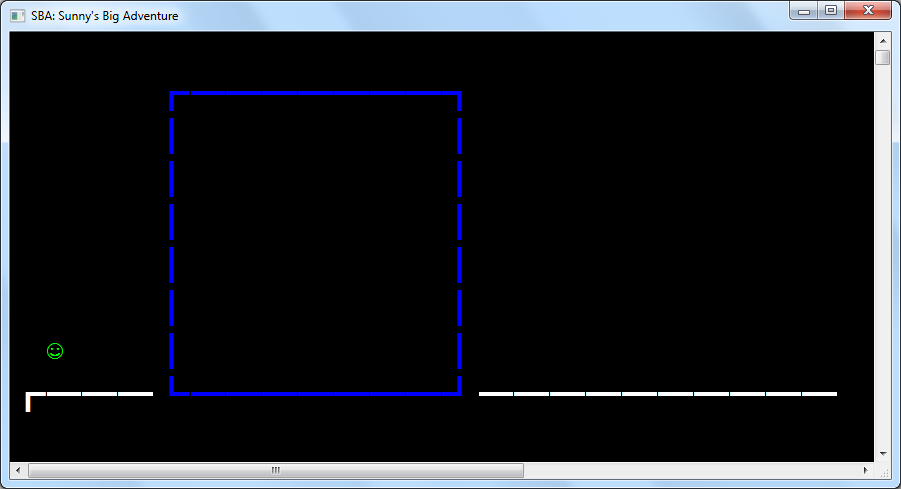
|  |
| --- |
| ' 3. Avoid white win  For Each x In choices  For Each x2 In choices  If WhoWin((x, x2)) = Player.Player Then avoidChoices.Add(x)  Next  Next |

If the player will not immediately win, the CPU will first filter away its choices that could lead to white winning.

|  |
| --- |
| ' 4. Random placement  choices.ExceptWith(avoidChoices)  Return If(choices.Count > 0, choices.RandomItem(),  If(avoidChoices.Count > 0, avoidChoices.RandomItem(), New Integer?())) |

It will then try to choose a column that is not filtered from the remaining columns randomly.

|  |
| --- |
| End Function  End Class |



**3.8.6: Region 5 – Win Screen**



|  |
| --- |
| Class Region5\_Win  Inherits Region  Protected ReadOnly Win As New TextEntity(WriteEntities, "You win!! Press Enter to start again.", New Point(6, 0))  Protected ReadOnly Fireworks As Sprite() = {Firework\_1, Firework\_2, Firework\_3} |

Fireworks is an array storing all possible firework sprites to choose from.

|  |
| --- |
| Protected ReadOnly Trigger As New TriggerZone(WriteEntities, New Rectangle(0, 0, WindowWidth, WindowHeight),  Function(key)  If key = ConsoleKey.Enter Then  SetCurrentRegion = Function() New Region1\_Title()  SaveRegion(CurrentRegion) |

If the enter key is pressed, then Sunny will be teleported to the Title Screen to start the game again.

|  |
| --- |
| End If  Return False  End Function,  Sub() If Not TickEvent.GetInvocationList().Where(Function(x) x.Target Is Me).Any() Then AddHandler Tick, AddressOf OnTick, |

If OnTick is not already registered in Tick, then it will be registered on player entry.

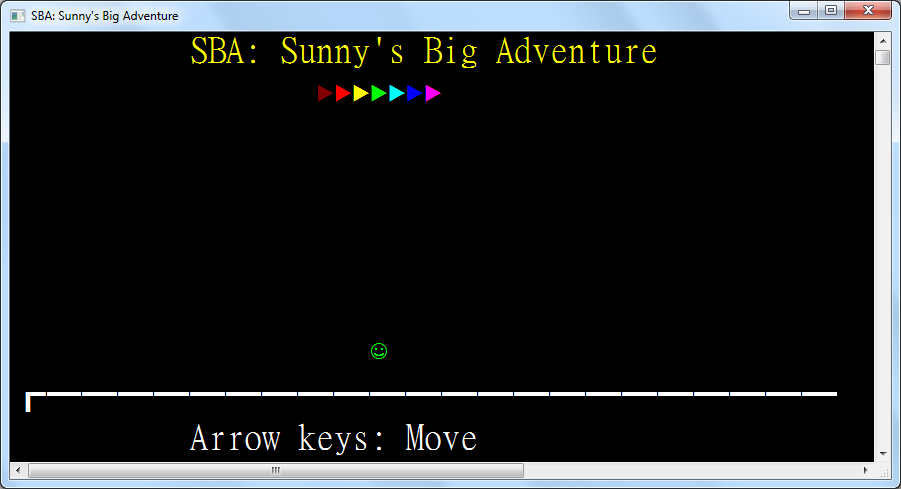
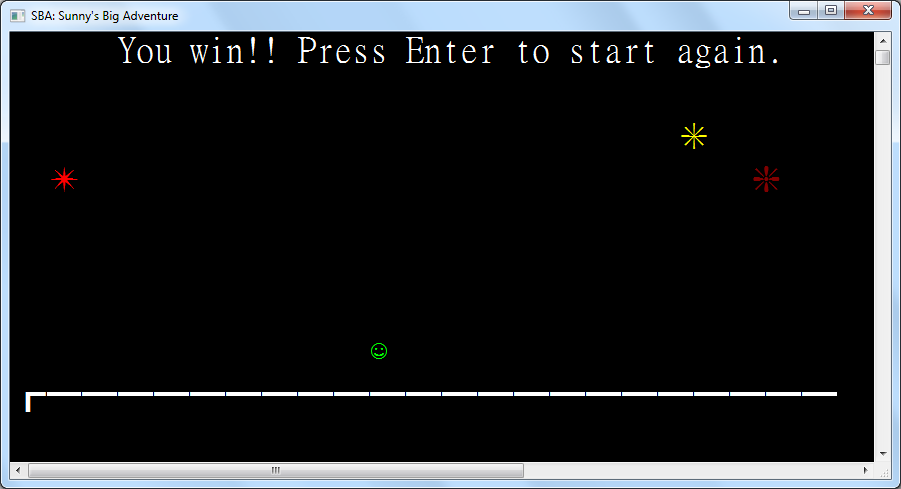
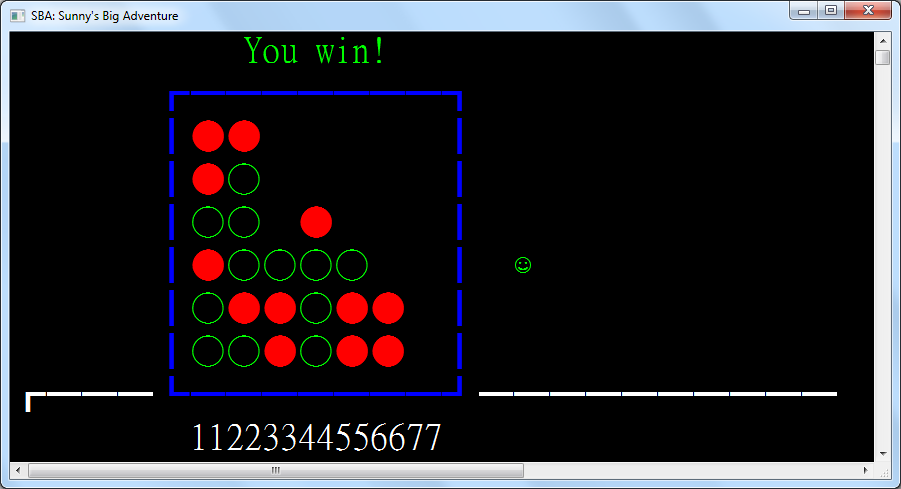
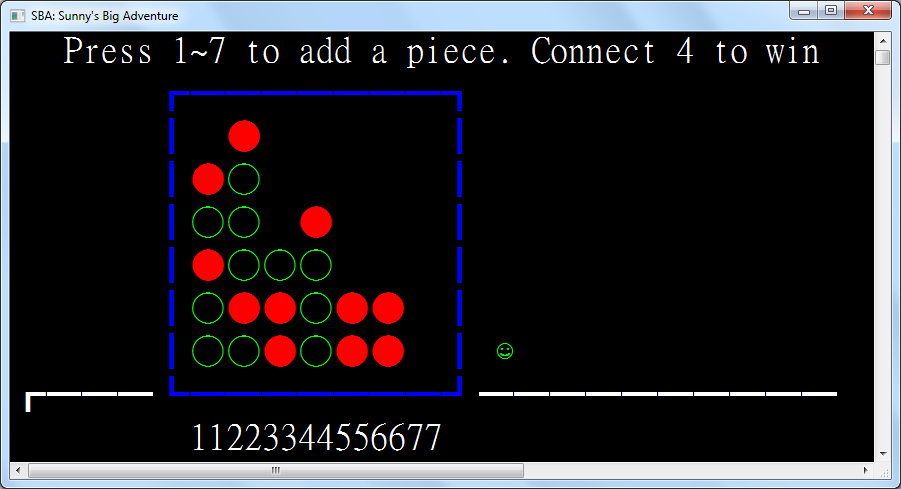
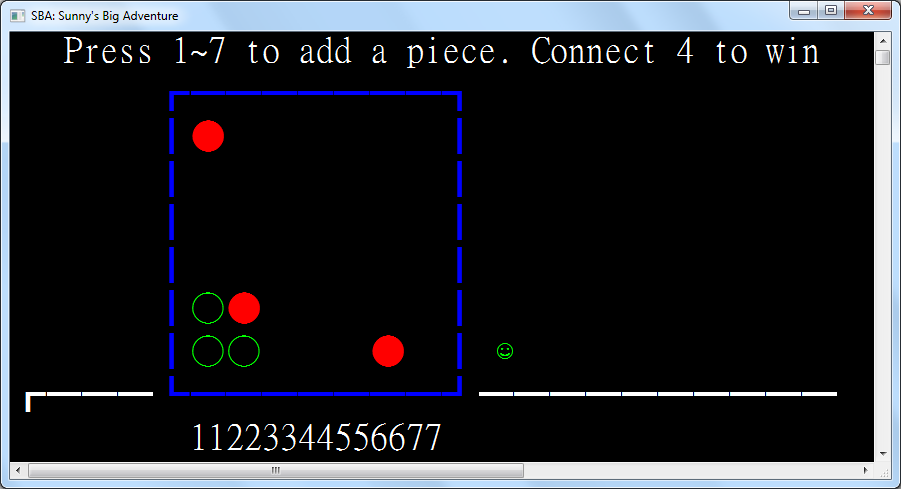
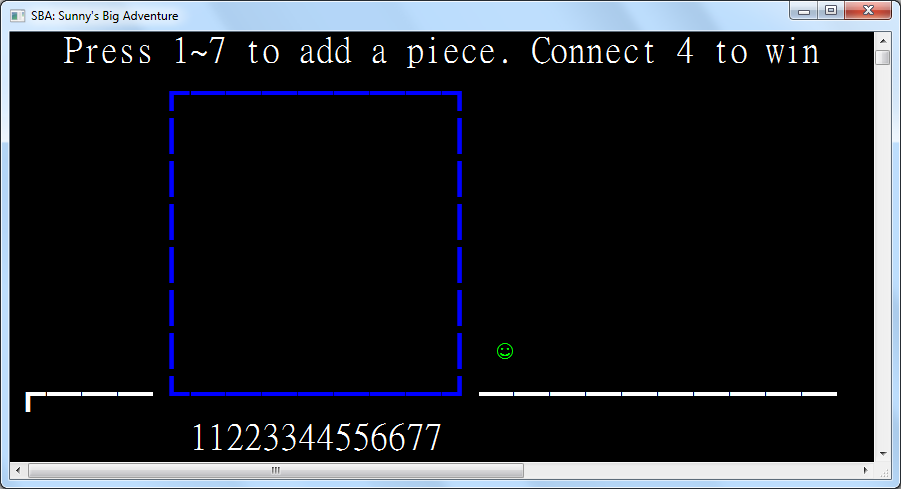
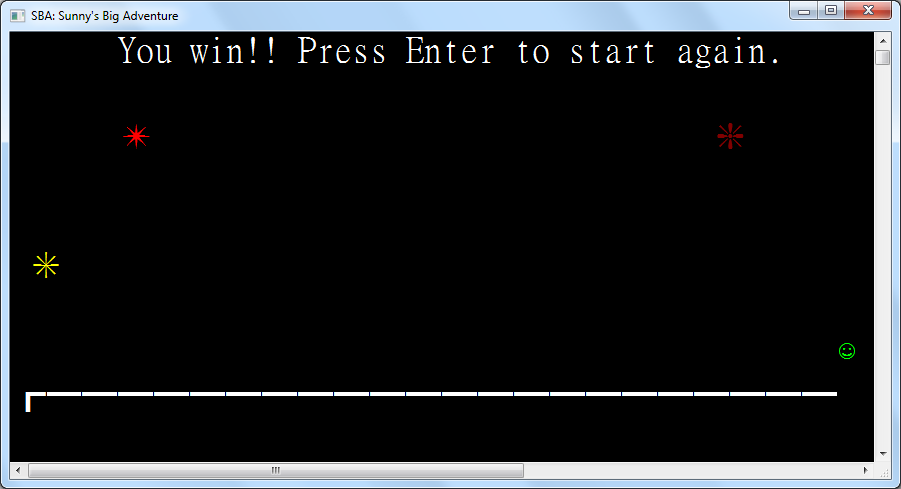
|  |
| --- |
| Sub() RemoveHandler Tick, AddressOf OnTick) |

When the player leaves the region, then OnTick will be removed from Tick.

|  |
| --- |
| Protected Sub OnTick()  WriteEntities.Add(New IteratingSpriteEntity(WriteEntities, Fireworks, New Point(Random.Next(WindowWidth), Random.Next(1, 7)))) |

In OnTick, a firework is spawned in a random position on the screen.

|  |
| --- |
| End Sub  Protected Overrides ReadOnly Property Left As Func(Of Region) = Function() New Region4\_ConnectFour()  Protected Overrides ReadOnly Property Right As Func(Of Region) = Nothing  End Class |



**3.9: Main program**



The implementation of the login and registration processes is 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.

|  |
| --- |
| Const WindowWidth = 48  Const WindowHeight = 10 |

As the window height and width are fixed, they are defined as constants for reuse across the program.

|  |
| --- |
| Public Event Tick()  Sub Main()  If LargestWindowWidth < 48 Or LargestWindowHeight < 10 Then  WriteLine("ERROR: Please decrease font size")  Return  End If |

The window being large enough is essential for the game to run correctly. This is reminded to the user to decrease the font size and thus increasing the available number of characters on the screen.

|  |
| --- |
| OutputEncoding = Encoding.Unicode  Title = "SBA: Sunny's Big Adventure"  Console.WindowWidth = WindowWidth  Console.WindowHeight = WindowHeight |

The console window is initialized to have an encoding that can display Unicode characters – the happy sprite and the angry sprite. The title is also set and the window size is set to constants as defined above.

|  |
| --- |
| If Not IO.File.Exists(FileName) Then IO.File.Create(FileName).Dispose() |

If the save file is not already created, it is created before the login screen is displayed.

|  |
| --- |
| Login: WriteLine("Login")  WriteLine("-----")  While \_currentRegion Is Nothing |

If the current region is null, this means that the user has not successfully logged in yet. The login loop continues.

|  |
| --- |
| Write("User name (Empty to register): ")  UserName = ReadLine()  If UserName = "" Then |

After asking for the username, it is compared to the empty string which indicates that the user wishes to register.

|  |
| --- |
| Clear()  WriteLine("Register")  WriteLine("--------")  While UserName = "" |

If the username is empty, the user has not successfully registered yet. The registration loop continues.

|  |
| --- |
| Write("User name (Empty to login): ")  UserName = ReadLine()  If UserName = "" Then |

After asking for the username, it is compared to the empty string which indicates that the user wishes to login.

|  |
| --- |
| Clear()  GoTo Login |

As Visual Basic does not have multi-level continue statements, a GoTo statement is used in its place.

|  |
| --- |
| Else  If IO.File.ReadLines(FileName).Any(Function(line) line.Split(FieldSeparator)(0) = UserName) Then  WriteLine("User already exists.")  UserName = "" |

A matching record in the data file being found after sequentially searching means that a user with a matching name has already registered and the new user has to choose a different username.

|  |
| --- |
| End If  End If  End While  Write("Password: ")  IO.File.AppendAllLines(FileName, {String.Join(FieldSeparator, UserName, ReadLine(), GetType(Region1\_Title).FullName)}) |

After entering a password to register, a new user record is appended to the data file.

|  |
| --- |
| WriteLine("Registration successful. Press Enter to start!")  ReadLine()  Clear()  \_currentRegion = New Region1\_Title |

After successful registration, the user can press enter to start a new game.

|  |
| --- |
| Else  Dim config = IO.File.ReadAllLines(FileName)  Dim line = config.Select(Function(l) l.Split(FieldSeparator)).FirstOrDefault(Function(l) l(0) = UserName)  If line Is Nothing Then  WriteLine("User not found.") |

When the user enters a username for logging in, the corresponding record is found by linear search and matching the records’ username fields with the provided username. If not found, then the user will be notified and the login loop continues.

|  |
| --- |
| Else  Write("Password: ")  If ReadLine() = line(1) Then  Clear()  \_currentRegion = CType(Type.GetType(line(2)).GetConstructor({}).Invoke({}), Region)  Else  WriteLine("Incorrect password.")  End If |

When the user enters a password, it will be compared to the corresponding password field of the record. If the password is correct, then the progress will be deserialized to a region object. If incorrect, then the user will be notified and the login loop continues.

|  |
| --- |
| End If  End If  End While  CursorVisible = False  Sunny.Position = New Point(2, 5) |

Upon successful loading of progress, the cursor will disappear and Sunny will appear in the loaded region.

|  |
| --- |
| While True  RaiseEvent Tick()  Dim key = ReadKey(TimeSpan.FromSeconds(0.2))  If key IsNot Nothing Then ActiveEntity.HandleKey(key.GetValueOrDefault())  End While |

Finally, the game loop will be entered where a tick is raised every 0.2 seconds or when the player enters a key.

|  |
| --- |
| End Sub |

**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.

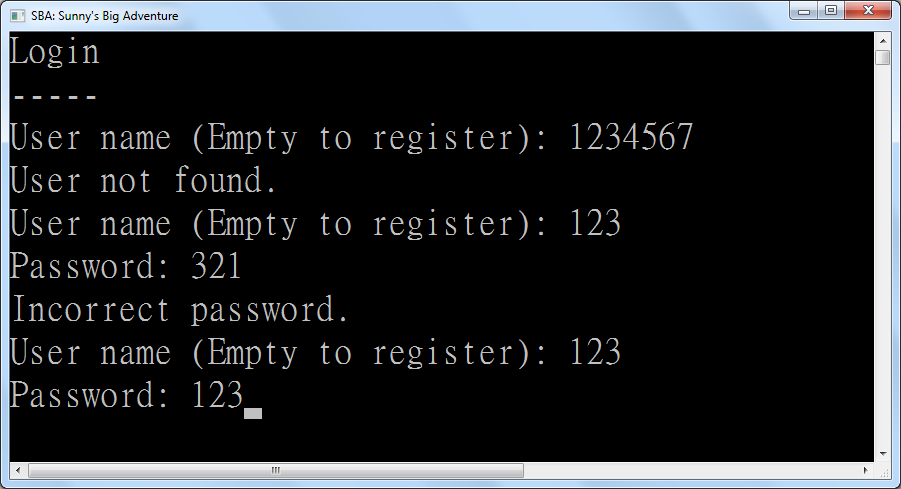
The following will be shown:

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

**4.2: Test Cases Design and Results**

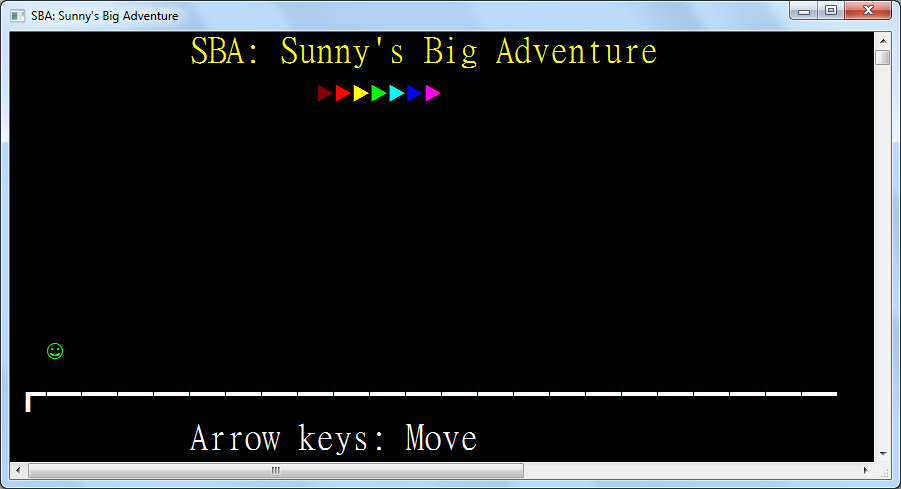
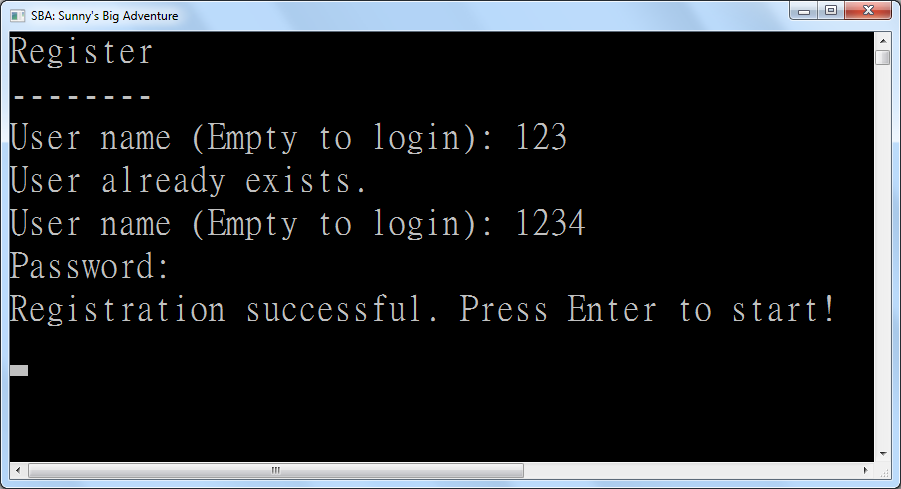
In this sub-chapter, testing is divided for each subprogram where test cases will first be described, then the expected outcome and the test results, and finally whether it matches the expected outcome and problems encountered if any. The details of the testing are as follows.

**4.2.1: Tests for Login**



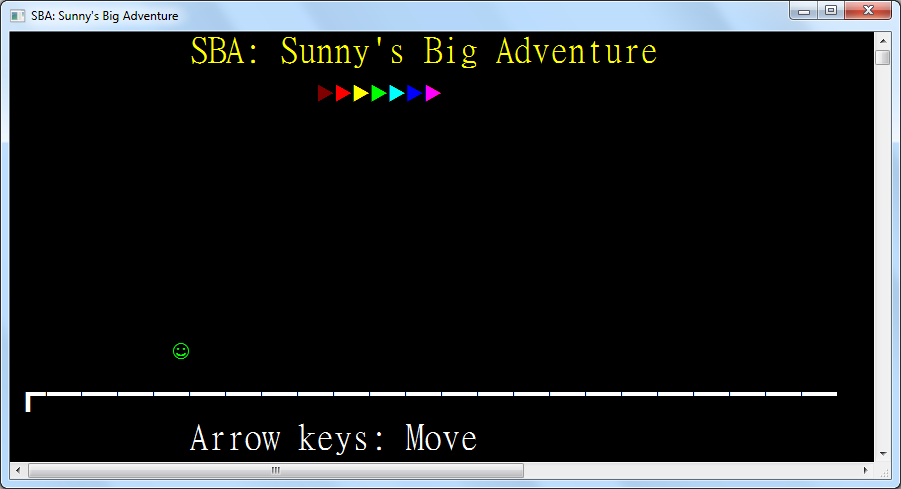
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Name: 1234567 | Invalid Input | Display “User not found” | As expected |
| Name: @#^786923gyuagd | Invalid Input | Display “User not found” | As expected |
| Name: 123  Pass: 321 | Invalid Input | Display “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**



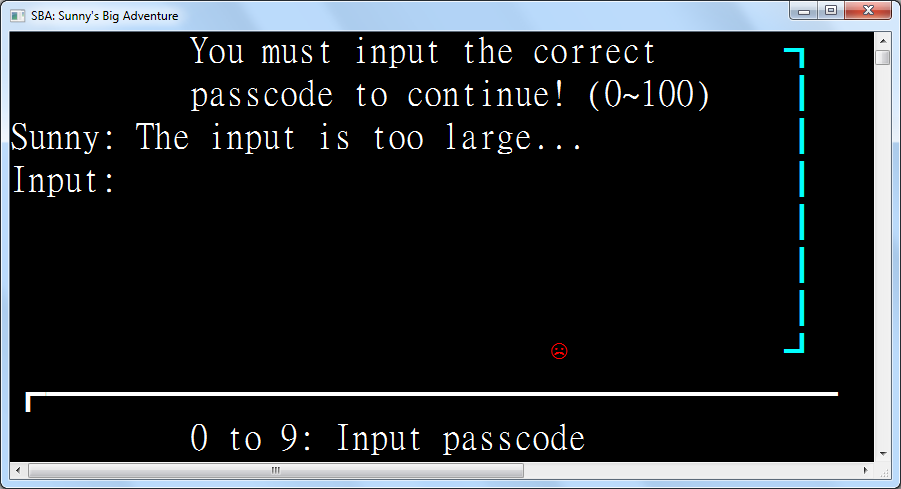
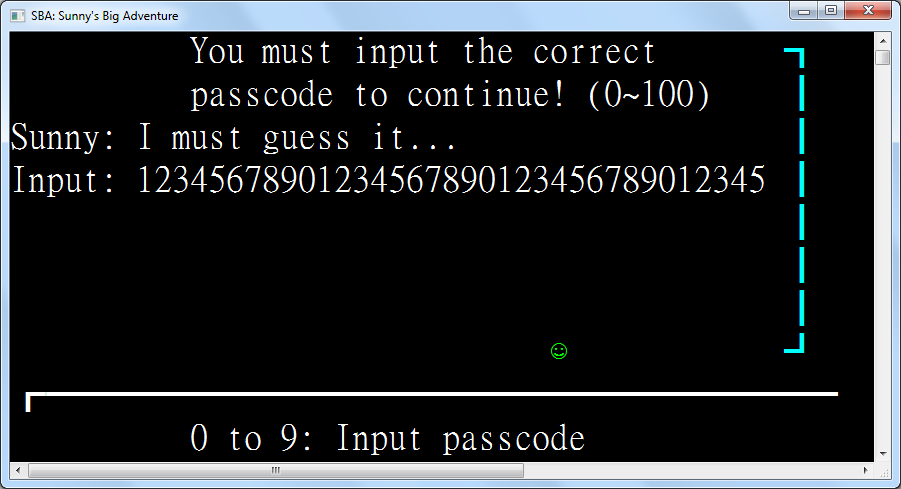
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Name: 123 | Invalid Input | Display “User already exists” | As expected |
| Name: 1234  Empty PW | Valid Input | Successfully register | As expected |
| Name: 12345  Pass: 12345 | Valid Input | Successfully 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**



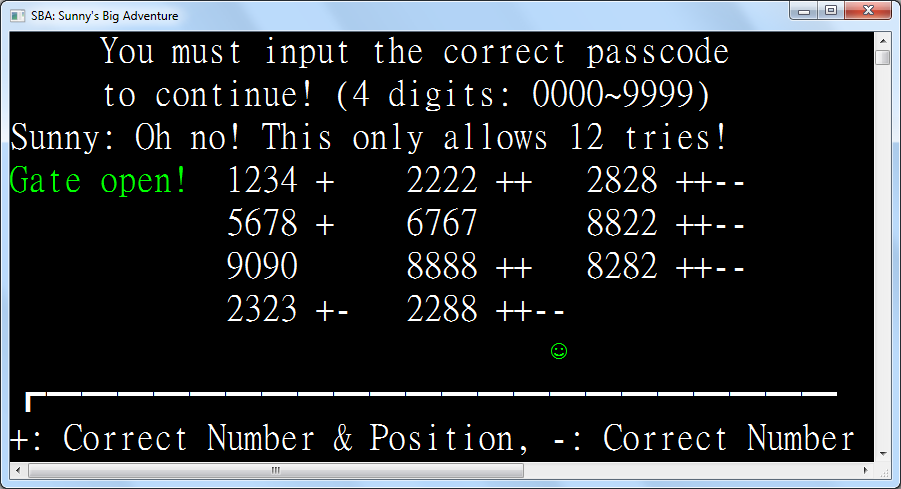
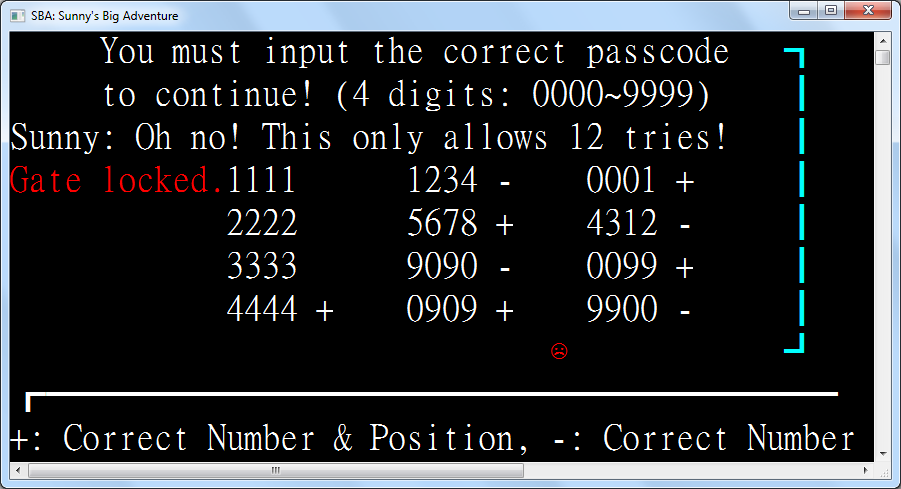
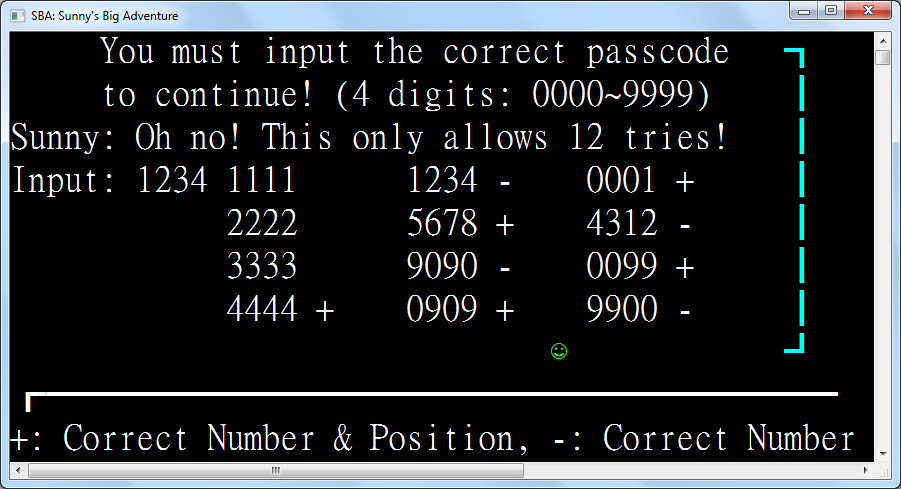
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| Left arrow when there is space to move to | Valid Input | Sunny moves left | As expected |
| Left arrow at the left boundary | Extreme Input | Nothing happens | As expected |
| Right arrow when there is space to move to | Valid Input | Sunny moves right | As expected |
| Right arrow at the right boundary | Extreme Input | Sunny moves to the next region | As expected |
| Up arrow on the ground | Valid Input | Sunny jumps and is pulled down by gravity | As expected |
| Up arrow mid-air | Invalid input | Nothing happens | As expected |
| Down arrow when avatar is in air | Valid Input | Sunny 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**



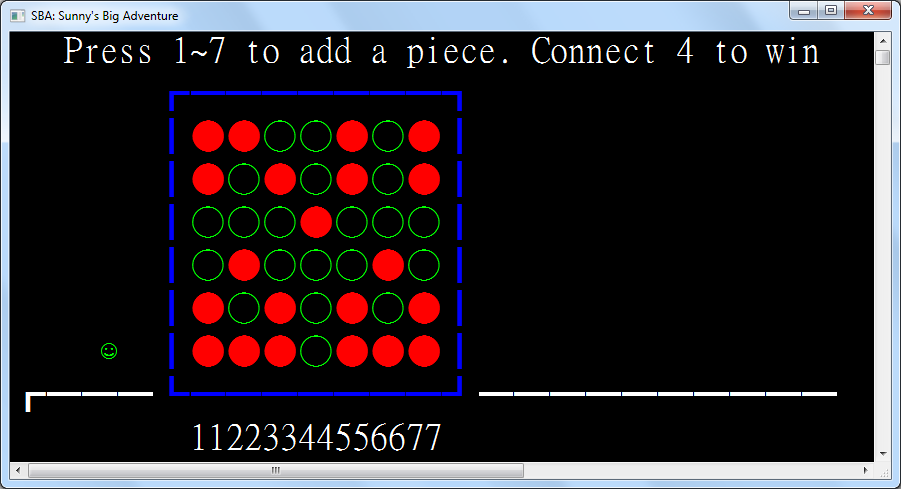
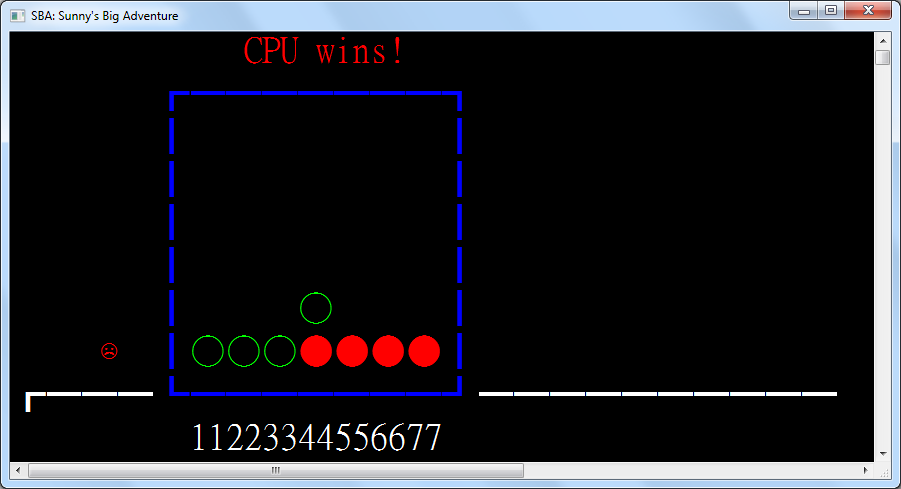
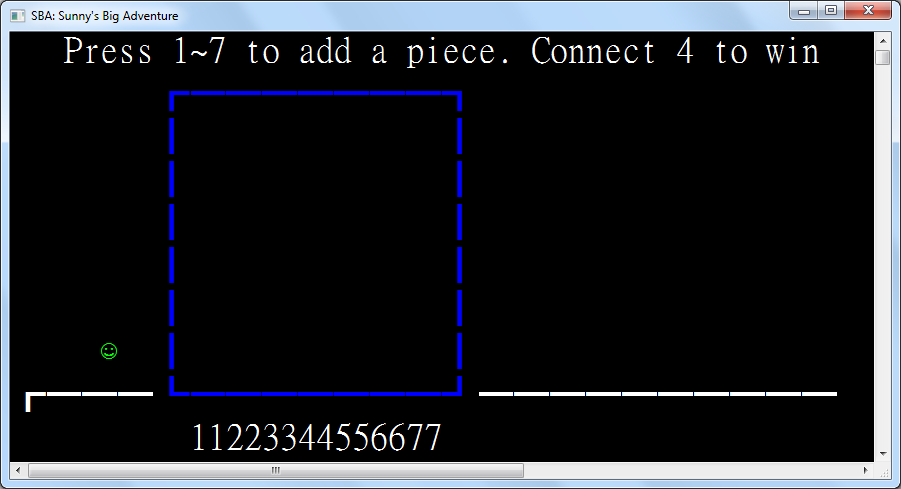
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| 12345678901234567890123456789012345 | Valid Input | Display “Input too large” | As expected |
| 0 | Valid Input | Display “Input too small” | As expected |
| -1 | Invalid Input | The minus sign is ignored | 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**



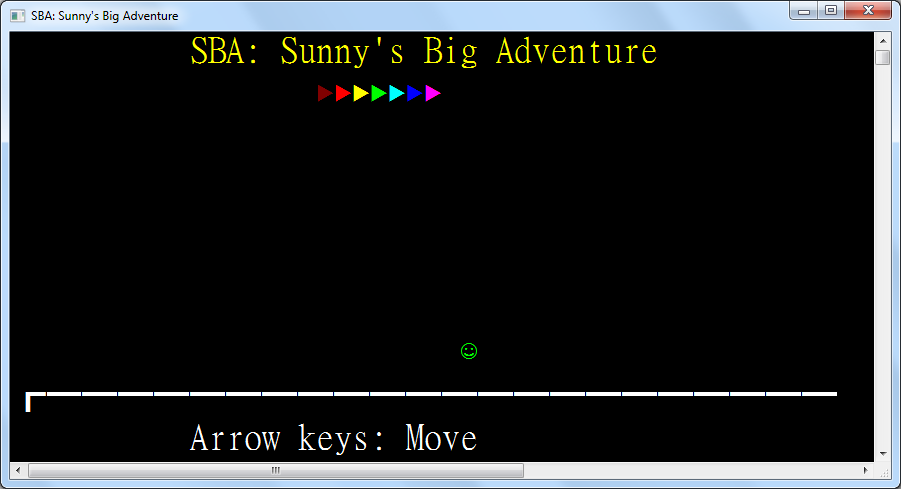
|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Type | Expected Outcome | Results |
| 1111 | Extreme Input | Display +s and –s depending on the digits | As expected |
| 1234 | Valid Input | Display +s and –s depending on the digits | As expected |
| 5678 | Valid Input | Display +s and –s depending 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 when there is space to move to | Valid Input | Sunny moves left | As expected |
| Left arrow at the left boundary | Extreme Input | Sunny moves to the previous region – Connect Four | As expected |
| Right arrow when there is space to move to | Valid Input | Sunny moves right | As expected |
| Right arrow at the right boundary | Extreme Input | Nothing happens | As expected |
| Up arrow on the ground | Valid Input | Sunny jumps and is pulled down by gravity | As expected |
| Up arrow mid-air | Invalid input | Nothing happens | As expected |
| Down arrow when avatar is in air | Valid Input | Sunny drops faster | As expected |
| Enter | Valid Input | Sunny 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 | Sunny is blocked from overlapping with the firework | As expected |

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

**Chapter 5:**