

# **PRISON BREAK**

## **SOFTWARE REQUIREMENT SPECIFICATION**

**19<sup>TH</sup> OCTOBER,2014**

**PRISON BREAK**

**PRATIK BABHULKAR**

**SHREERANG KAORE**

**AMIT PATIL**

**NITIN CHOUDHARY**

**CS101 PROJECT**

**IIT BOMBAY**

## Team details:

<b>PRATIK BABHULKAR</b>	140100013	CS101 GROUP 9
<b>SHREERANG KAORE</b>	140100028	CS101 GROUP 9
<b>AMIT PATIL</b>	140100043	CS101 GROUP 9
<b>NITIN CHOUDHARY</b>	140100048	CS101 GROUP 9

## Software requirements:

- We need to have CODE BLOCKS integrated development environment installed on the pc with operating system windows 7 or above/ubuntu 12.04 and above.
- Then we need to have simplecpp graphics package installed on the pc and integrated with the CODE BLOCKS.
- Simply open the game with code blocks and play !

# PRISONBREAK

## INTRODUCTION:

The aim of the project is to develop a game named "PRISON BREAK". This document explains in detail the game, certain technical specifications involved in its implementation, algorithm for creating the game.

The game is sort of copy of the game "UNBLOCK ME". It will provide the player with a brain twisting experience. The aim of the player is to rescue or unblock the prisoner from prison by making a way for it through the hole or gate.

## TECHNICAL SPECIFICATIONS:

The whole software will be developed using CODE BLOCKS integrated development environment and SIMPLECPP for the graphics. We have main two algorithms.

- 1) **Creating the levels randomly**
- 2) **Playing the game.**

The program goes like, user will decide to play a difficulty level and 1<sup>st</sup> algorithm will create a level of that difficulty. Then the corresponding arrangement of blockers is passed to second algorithm and level is played. Again the player is asked to tell which level he/she wants to play and game runs accordingly.

Approximate algorithms are given in subsequent pages of the document.

# ALGORITHM for creating Random Arrangements of Blockers:

## 1) Function for creating the board (Prison) (This function needs to be called once)

### **create\_prison\_func**

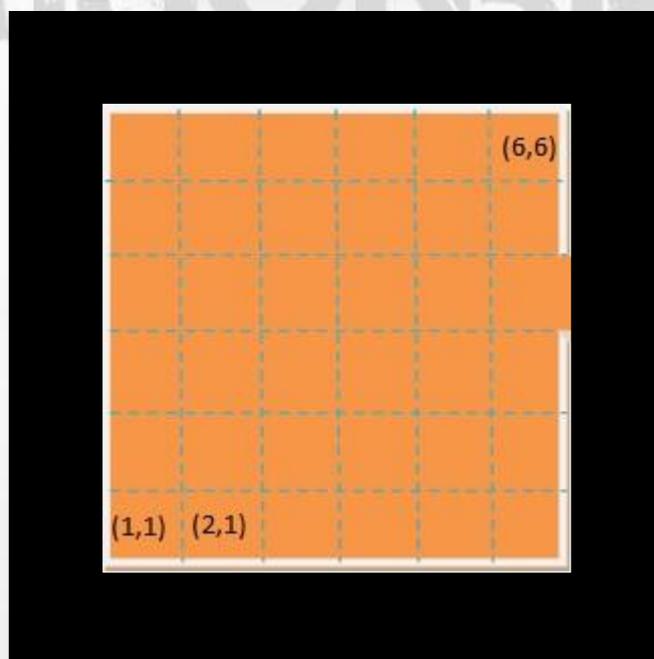
**Precondition :** \*\* take the theme as an integer input\*\*

\*\* Call the function on click on start game \*\*

**Post condition :** \*\* Create and display the prison with exit in colour of the theme\*\*

**Algorithm for the function :**

- Define a square size say  $m \times m$ . (say This is my unit square A)
- Create a square of size such that it can contain 6 rows and columns of A.
- Define co-ordinates in terms of A's. (e.g (1,1) refers to A in bottom left corner and (6,6) refers to A in top right corner)
- TO make an exit, delete the right border of board next to (6,4)
- Sample board is given. The lines are just shown to represent that the board consists of  $6 \times 6$  A.



## 2) Function for creating a random arrangement of blockers and prisoner in the prison for game.

### **random\_arrangement\_func**

**Preconditions :** \*\*Take input value of difficulty level D\*\*

**\*\*D should be integer value from 1 to 100\*\***

**\*\*Take theme as an integer input varying from 1-3\*\***

**Post conditions :** **\*\*Output a random configuration of blockers and prisoner in the prison in terms of their co-ordinates, lengths, colours(according to theme)\*\***

**Algorithm for the function:**

- Take Difficulty level (say D) as an integer parameter .
- Say number of blocks =num\_block and number of minimum moves to win is num\_moves.
- Define num\_block= Random number between
  - (4-6 if D is 01-10)
  - (7-8 if D is 11-25)
  - (9-10 if D is 26-50)
  - (11-13 if D>50)

{We might change this at a later stage.}

- Define num\_moves=5 if  $0 < D < 8$ 
  - =6 if  $7 < D < 15$
  - =7 if  $14 < D < 22$
  - =8 if  $21 < D < 30$
  - =9 if  $29 < D < 35$
  - =10 if  $34 < D < 50$
  - =11 if  $49 < D < 100$

(Number of minimum moves increases as the difficulty level increases)

- Define a structure **BLOCK** containing all information about the block like
  - 1) Boolean variable whether block is horizontal or vertical
  - 2) Length of block
  - 3) X- co-ordinate of lower end(if vertical) or left end (if horizontal) ,
  - 4) y-co-ordinate of lower end(if vertical) or left end (if horizontal),
  - 5) Colour of Block
- For our main block (prisoner) define **BLOCK** variable **prisoner** as follows:
  - 1) Horizontal
  - 2) Length=2
  - 3) X-co-ordinate=random integer from 1-4
  - 4) Y-co-ordinate=4
  - 5) Colour=RED
- Define an array **BLOCKERS** of **BLOCK** type with size **num\_blocks**.
- **BLOCKERS []=**
  - {
    - 1) Horizontal and vertical alternately
    - 2) Length : if  $6 > D$  num\_blocks/2 are of length 2
      - If  $11 > D > 5$  , num\_blocks/2 +1 are of length 3
      - If  $18 > D > 10$ , num\_blocks/2 are of length 2
      - If  $26 > D > 17$ , num\_blocks/2 +1 are of length 3
      - If  $36 > D > 25$ , num\_blocks/2 are of length 2

If  $51 > D > 35$ ,  $\text{num\_blocks}/2 + 1$  are of length 3.

If  $d > 50$ , 5 blocks of length 3

3) Assign the x coordinates randomly but taking care of non-overlap of Blockers

4) Same for Y co-ordinates

5) Colour=according to theme (We will define it later in stage 2)

}

- Call the **count\_min\_steps** by passing **BLOCKERS** and other info about blocks.
- If min steps not equal to **num\_steps**, call **random\_arrangement\_func** with same **theme** and **D**.
- Else if min steps are equal to **num\_steps**, return.

### 3) Function for counting minimum number of steps as an integer .

#### **count\_min\_steps**

**Preconditions :** \*\*Take the array including all information of blockers and prisoner\*\*

**Post conditions:** \*\*Return no. of minimum steps to unblock the prisoner if less than 12\*\*

\*\*\*Return -1 indicating many number of steps\*\*

#### **Algorithm for function:**

We are using following algorithm for counting minimum number of steps.

- Take the initial configuration or random board (Prison) from the function.
- Maintain a running integer variable initialised to 1.
- Check whether my prisoner is already unblocked. If so return that integer. Else increment the integer.
- Do all the possible moves and temporarily save all the configurations.
- Check whether my prisoner is unblocked in atleast one of the configurations. If so return the integer. Else increment the integer.
- Again create all the possible configurations except those which are already formed. If the configuration is already formed earlier, discard it.
- Then check each of the configuration same way and continue till running integer=12.
- If Running integer=12 , return to the **random\_arrangement\_func**
- If prisoner is escaped at any stage, then check whether running integer = D . If so, return to play the game, else return to the **random\_arrangement\_func**.

### 4) Main function

#### **main\_func**

- Display themes and tell to select in terms of integer.
- Take input as an integer and assign it to variable **theme**.
- Take input an integer between 1-100 and assign it to **D**.
- Then call the **create\_prison\_func** by passing **theme** .
- Then call the **random\_arrangement\_func** by passing **theme** and **D**.
- Return to play game.

# ALGORITHM for Playing the game:

## 1) Main function

- Call **create\_prison\_func** .
- Repeat  
{Call other algorithm with parameter **theme** and difficulty level **D**.
- Take the arrangement of blockers, prisoner from other algorithm.
- Draw the rectangles representing the blockers and prisoner.
- Do  
{  
Repeat  
{Get the co-ordinates **x\_click , y\_click** of click by user.

Locate the blocker or prisoner which is being clicked.

Repeat

{Then take the input as arrow key as left,right,up,down.

If selected block is horizontal

{

if input is up or down {do nothing}

If input is left {check if there is free space on left.

If there is free space {decrease x co ordinate by 1.}}

If input is right {check if there is free space on right.

If there is free space{increase x-co-ordinate by 1}}

}

If selected block is vertical

{

If input is left or right {do nothing}

If input is up {check if there is free space on upper.

If there is free space{increase y co-ordinate by 1.}}

If input is down {check if there is free space below.

If there is free space{decrease y-co-ordinate by 1.}}

```
}
```

```
If click is received {break out of this loop}
```

```
}
```

```
}}
```

Ask which difficulty level you want to play next

# PRISONBREAK

## Some other important things:

The algorithm for generating the random arrangement has a problem. It may take long time.

We are taking **D** as difficulty level and then randomly creating the prison and arrangement. But to be sure that it is of the difficulty level **D**, we are solving the game or actually unblocking the prisoner in the function **count\_min\_steps**. In this function if the number of minimum steps is not how much we require (**num\_steps**), then we repeat the process. This may be time consuming. So we are working on some other algorithm to define the difficulty level of the game.

Till that we have decided that, we can also have a different solution. We can actually create different random arrangements using **random\_arrangement\_func**, then calculate the difficulty level by using function **count\_min\_steps**. Suppose if the difficulty level comes out to be 3, we save that arrangement as the difficulty level 3 and regenerate another arrangement and again calculate its level and save it. This way, we can save 100 randomly generated levels and use them in playing as if they are pre-defined levels directly.

## References :

Till now we have accessed information from following sources and seek to have more info and work.

- Introduction to problem solving and programming through C++ by **ABHIRAM RANADE** for simplecpp
- <http://users.softlab.ece.ntua.gr/~ttsiod/unblock.html> for things like auto solving.
- <http://stackoverflow.com/questions/12236187/algorithm-to-generate-a-random-board-in-the-game-unblock-me>
- <https://www.youtube.com/watch?v=naXUIEAlt4U> for understanding random number generator in C++.
- Previous years' projects.