ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH


# KIẾN TRÚC MÁY TÍNH - MỞ RỘNG (CO200D)

## Bài tập lớn (Học kỳ 221)

# Connect Four

**Advisors**: Phạm Quốc Cường
Trần Thanh Bình
Nguyễn Xuân Minh

**Students**:

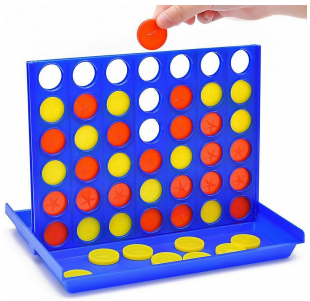
| | |
|---|---|
| Nguyễn Thái Tân | 2112256 |
| Trần Thiện Nhân | 2111913 |
| Mai Hoàng Danh | 2110896 |
| Nguyễn Phúc Minh Quân | 2110479 |

Ho Chi Minh City, 12/2022

# Mục lục

# 1 Connect Four

Connect Four là một biến thể của cờ Caro, phiên bản 3D và thú vị hơn nhiều. Bàn cờ được đặt thẳng đứng theo chiều dọc. Người chơi thả các quân cờ vào một trong 7 hàng dọc, mỗi hàng chứa được 6 quân. Người nào đạt được 4 quân cờ nối liền nhau theo chiều ngang, dọc hoặc chéo sẽ chiến thắng.



# 2 Thuật toán

Ở phạm vi bài tập lớn này, do việc hiện thực AI cho máy tính có thể chơi đối kháng với người chơi là rất phức tạp, nhóm quyết định sẽ hiện thực trò chơi với hướng thiết kế là Multiplayer Game, tức là chơi giữa 2 người.

## 2.1 Ý tưởng

Ta sẽ chọn một mảng hai chiều (ma trận) để hiện thực trò chơi với quy ước: giá trị tại một vị trí bằng

- $0$ nếu vị trí đó chưa được ai đánh dấu,
- $1$ nếu vị trí đó được người chơi 1 đánh dấu,
- $2$ nếu vị trí đó được người chơi 2 đánh dấu.

Về cơ bản, một mảng hai chiều sẽ được lưu trên bộ nhớ với cấu trúc tương tự như mảng một chiều. Vì vậy, ta sẽ "trải phẳng" mảng hai chiều $6 \times 7$ đã nói trên để hiện thực giống như mảng một chiều.

Khi đó, để truy cập mảng hai chiều tại vị trí hàng $i$, cột $j$, ta truy cập mảng một chiều tại vị trí $i \times 7 + j$.

Vì một vị trí trong mảng, ta chỉ cần các giá trị 0, 1, 2 nên ta chỉ cần $1$ *byte* cho một vị trí. Do đó ta sẽ hiện thực một mảng kiểu *byte* có kích thước $7 \times 6 = 42$ *byte*. Ta sẽ gọi mảng này là *BoardArray*

## 2.2 Thuật toán chung

Bước 1: Tạo một ma trận với kích thước $6 \times 7$, tất cả đều có giá trị bằng $0$.

Bước 2: Người chơi 1 sẽ nhập vào số thứ tự (từ $0$ đến $6$) của cột mình muốn *thả vòng tròn* vào.

Bước 3: Thay đổi giá trị tại vị trí vừa *thả vòng tròn*. Kiểm tra điều kiện thắng. Nếu thắng hoặc hòa, dừng thuật toán.

Bước 4: Người chơi 2 sẽ nhập vào số thự tự (từ 0 đến 6) của cột mình muốn *thả vòng tròn* vào.

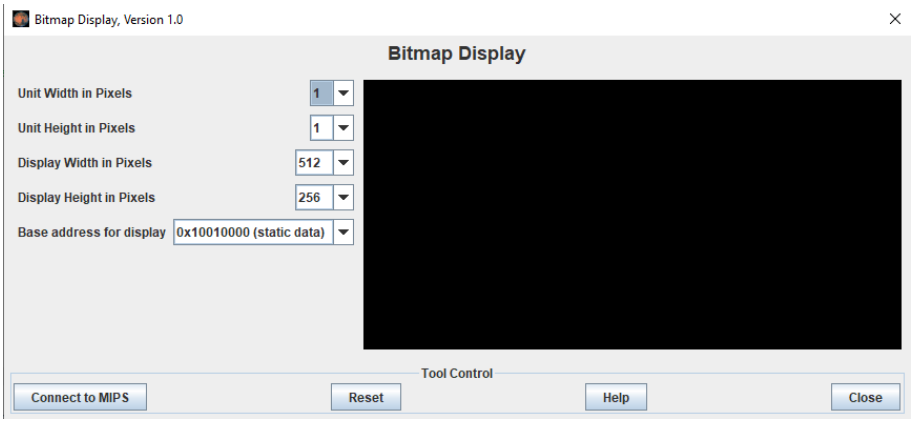
Bước 5: Thay đổi giá trị tại vị trí vừa *thả vòng tròn*. Kiểm tra điều kiện thắng. Nếu thắng hoặc hòa, dừng trò chơi.

Bước 6: Quay lại bước 2.

**Thuật toán Kiểm tra điều kiện thắng**
Về thuật toán này, chúng ta sẽ tìm hiểu kĩ hơn ở phần sau.

# 3 Bitmap Display

Bitmap Display là công cụ hỗ trợ có sẵn trong MARS MIPS, nó giúp chúng ta biểu diễn hình ảnh một cách trực quan. Nó không hiển thị nội dung của bộ nhớ dưới dạng bitmap mà thay vào đó, nó vẽ sơ đồ pixel dựa vào bộ nhớ nơi đặt bộ đệm hiển thị.




Giao diện của Bitmap Display như sau:

- Unit Width/Height in Pixels: Độ rộng/cao của một *Unit* tính theo Pixel (có các lựa chọn 1/2/4/8/16/32)
- Display Width/Height in Pixels: Độ rộng/cao tính theo Pixel (có các lựa chọn 64/128/256/512)
- Base address for display: (có các lựa chọn 0x10000000 global data / 0x10008000 $gp / 0x10010000 static data / 0x10040000 heap / 0xffff0000 memory map)

Với các thông số như trên, ta có chiến lược hiện thực trên Bitmap Display như sau:

- Vì độ phân giải của màn hình thông thường là $1366 \times 768$, nên chúng ta sẽ chọn kích thước của Bitmap Display là $512 \times 512$.
- Một hàng có 7 cột (7 *vòng tròn*), cộng thêm 8 *viền* của khung, coi như một hàng có 8 cột. Kết hợp với nhận xét vừa nêu, chúng ta chọn một *vòng tròn* sẽ có độ rộng là $512/8 = \mathbf{64\ pixel}$, một *viền* sẽ có độ rộng $512/8/8 = \mathbf{8\ pixel}$.
- Do một *vòng tròn* và một *viền* đều có độ rộng chia hết cho 8, ta sẽ chọn $\mathbf{1\ \textit{Unit} = 8\ pixel}$.
- Hiển nhiên, *vòng tròn* có độ rộng phải bằng độ cao. Khi đó độ cao của bảng sẽ là $64 \times 6 + 64 = 448$ pixel, nghĩa là vẫn còn phần dư ở ngoài bảng. Ta sẽ tận dụng phần dư đó để viết các số của cột.

- Ta sẽ chọn load địa chỉ Base address trên heap.

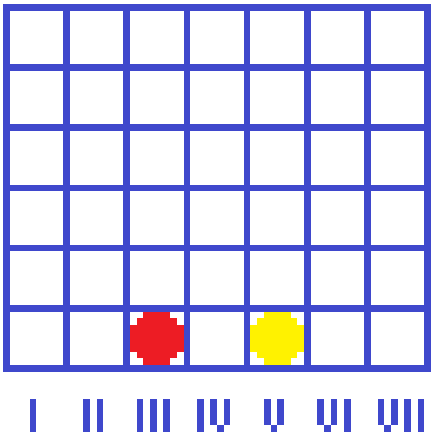
Bảng màu được dùng:

| Màu | Mã HEX | Dùng để |
|---|---|---|
| Màu xanh dương | 0x0000FF | Vẽ khung viền |
| Màu đỏ | 0xFF0000 | Vẽ vòng tròn của người chơi 1 |
| Màu vàng | 0xE5C420 | Vẽ vòng tròn của người chơi 2 |
| Màu trắng | 0xFFFFFF | Vẽ khung trống |

# 4 Hiện thực trên MARS MIPS

## 4.1 Các hiện thực liên quan đến việc vẽ trên Bitmap Display

Với hướng xây dựng bên trên, ta phác thảo trước một bảng như sau:



Đến đây, ta có thêm một vài nhận xét:

- Ta cần vẽ rất nhiều *đường thẳng đứng* và *đường gạch ngang*. Tuy nhiên, chúng có những độ dài khác nhau (ví dụ như *đường thẳng đứng* của khung có độ dài 55, còn của ký tự I của số la mã chỉ có độ dài 5).
- Hơn nữa, thật ra *vòng tròn* cũng chỉ là những *đường gạch ngang* có độ dài khác nhau.
- Vì vậy, ta sẽ hiện thực một thuật toán để vẽ hai loại đường đó với độ dài khác nhau. Tức là tương tự như một hàm *void DrawVerticalWithLength(int x, int y, int color, int length)*, với $x, y$ là vị trí tọa độ trong Bitmap Display còn *color* và *length* lần lượt là màu và độ dài mà ta muốn vẽ.

**Hiện thực vẽ bảng**

Bước 1 Vẽ toàn bộ màu trắng lên Bitmap Display

Bước 2 Vẽ 7 đường gạch ngang màu xanh

Bước 3 Vẽ 7 đường thẳng đứng màu xanh

Bước 4 Vẽ các con số

Ta sẽ đi vào chi tiết hơn

### 4.1.1 Hiện thực Vẽ điểm trên Bitmap Display

Đây là bước hiện thực cơ bản và rất quan trọng trong bài làm. Mọi hiện thực liên quan đến việc vẽ lên màn hình Bitmap Display, ta đều phải dùng phương thức này.

Về cơ bản, Bitmap Display lấy dữ liệu từ Heap (như ta đã chọn) để tự động hiện ra màn hình. Do đó để biểu diễn các Unit lên Bitmap Display, ta thay đổi giá trị của Heap thành các mã màu tương ứng.

Với mỗi cặp vị trí $(x, y)$ trên bảng $64 \times 64$ unit, thì địa chỉ cần thay đổi giá trị trong Heap là $0x10040000 + x * 4 + y * 64 * 4$. Sau đó chỉ cần load giá trị của màu vào địa chỉ đó.

### 4.1.2 Hiện thực Vẽ đường gạch ngang / đường thẳng đứng

Để vẽ được một đường với độ dài *length*, chúng ta cần khởi tạo một biến $counter = length$, giảm dần sau mỗi vòng lặp để vẽ với thuật toán Vẽ điểm ở trên. Khi $counter = 0$, tức là ta đã vẽ xong, kết thúc thuật toán.

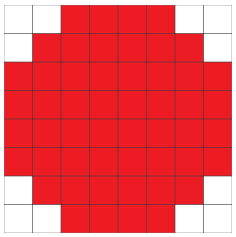
### 4.1.3 Hiện thực Vẽ bảng màu trắng

Một cách tự nhiên, ta sẽ vẽ tất cả 64 đường gạch ngang với màu trắng. Chỉ cần sử dụng một vòng lặp.

### 4.1.4 Hiện thực Vẽ *vòng tròn*

Như đã nói, một *vòng tròn* sẽ có kích thước theo Unit là 8 x 8. Tuy nhiên, ta chỉ vẽ lên một phần trong số đó để có một *vòng tròn* hợp lý.

Cụ thể, ta sẽ phác thảo hướng vẽ như sau:



Ở phần nhận xét trên, ta đã có chiến lược vẽ bằng các đường gạch ngang với độ dài khác nhau. Vì thế ta lưu trước một mảng hỗ trợ cho việc này.

$$CircleArray : .word\ 2, 4, 1, 6, 0, 8, 0, 8, 0, 8, 0, 8, 1, 6, 2, 4$$

Các cặp phần tử của mảng trên biểu diễn cho vị trí bắt đầu vẽ trên hàng và độ dài được vẽ của đường gạch ngang. Ví dụ, cặp đầu tiên $(2, 4)$ tức là ta sẽ vẽ từ vị trí thứ 2, vẽ 4 ô. Sau đó, ta dịch chuyển xuống 1 hàng, rồi tiếp tục cho đến khi duyệt hết mảng *CircleArray*.

### 4.1.5 Hiện thực Vẽ các số

Vì các số gần như không có quy luật cụ thể như vòng tròn ở trên, hơn nữa ta chỉ cần vẽ 7 số, nên ta quyết định sẽ vẽ "chay", tức là vẽ từng số.

Với ký tự I, chỉ đơn giản là ta sẽ gọi lại phương thức Vẽ đường thẳng đứng với một độ dài ngắn hơn, cụ thể trong trường hợp này là 5, bắt đầu từ các vị trí $(i, 59)$ cần thiết.

Với ký tự V, ta sẽ tận dụng một lần nữa phương thức Vẽ đường thẳng đứng. Ta sẽ vẽ hai đường thẳng đứng ở các cặp vị trí cần thiết $(i, 59)$ và $(i + 2, 59)$ với độ dài 4. Sau đó vẽ thêm chỉ một đường thẳng đứng với độ dài 1, tại ô $(i + 1, 63)$

## 4.2 Các hiện thực liên quan đến Gameplay

### 4.2.1 Hiện thực Nhập và Lưu chỉ số trên BoardArray

Bước 1 Người chơi sẽ nhập vào một số tự nhiên $x$ trên đoạn từ 1 đến 7. Nếu nhập vào ngoài đoạn thì sẽ thông báo lỗi và quay lại bước nhập.

Bước 2 Kiểm tra $BoardArray$ tại vị trí $x$

- Nếu $x > 41$, thông báo cột đã đầy, quay lại đầu bước 1.
- Nếu $BoardArray\,[x] == 0$, thì cập nhật $BoardArray\,[x]$ thành 1 hoặc 2 tùy vào lượt chơi. Kết thúc thuật toán.
- Nếu $BoardArray\,[x]! = 0$, thì ta phải duyệt ô nằm phía trên, tức là cập nhật $x = x + 7$. Quay lại đầu bước 2.

### 4.2.2 Hiện thực Kiểm tra điều kiện thắng

Từ vị trí vừa đặt vòng tròn vào, chúng ta sẽ phải kiểm tra tất cả 4 trường hợp sau: hàng ngang, hàng dọc, dấu gạch chéo lên ( / ), dấu gạch chéo phải ( \ ).

Với mỗi trường hợp:

Bước 1 $counter = 0, i = x$

Bước 2 Từ vị trí $x$ vừa đặt vòng tròn, duyệt về một phía (tăng $i$) sao cho ô được duyệt vẫn còn nằm trong *phạm vi* cho phép và cùng màu với vị trí $x$, ta được một vị trí $y$.

Bước 3 Từ $y$ ta lại duyệt về phía ngược lại (giảm $i$), sao cho ô được duyệt vẫn còn nằm trong *phạm vi* cho phép và cùng màu với vị trí $x$. Sau mỗi lần duyệt thì $counter = counter + 1$.

Bước 4 Đã duyệt xong. Nếu $counter > 3$ thì trò chơi kết thúc.

Bước 5 Kết thúc thuật toán

Cụ thể về *phạm vi* trong thuật toán trên:

- Hàng ngang: Ta chỉ duyệt hàng ngang tối đa từ $x - (x\%7)$ đến $x - (x\%7) + 6$. Với mỗi lần lặp, $i \to i \pm 1$. (Dấu $+$ khi duyệt lượt đi, dấu $-$ khi duyệt lượt về)
- Hàng dọc: Ta chỉ duyệt hàng dọc tối đa từ 0 đến 41. Với mỗi lần lặp, $i \to i \pm 7$.
- Dấu gạch chéo lên: Trường hợp này phức tạp hơn so với hàng dọc/ngang, vì phạm vi lần này được giới hạn bởi cả 4 đường (trong khi hàng dọc/ngang chỉ giới hạn bởi 2 hàng/cột ngoài cùng). Do đó khi duyệt, ta sẽ duyệt về hướng phải trên, cho đến khi $x > 41$ hoặc $x\%7 == 6$. Với mỗi lần lặp, $i \to i \pm 8$.
- Dấu gạch chéo phải: Tương tự với trường hợp dấu gạch chéo lên, khi duyệt, ta sẽ duyệt về hướng trái trên, cho đến khi $x > 41$ hoặc $x\%7 == 0$. Với mỗi lần lặp, $i \to i \pm 6$.

Sau khi kiểm tra 4 trường hợp, ta cũng cần phải kiểm tra ván đấu có hòa hay không. Với thao tác kiểm tra hòa, ta biết rằng ván đấu chỉ hòa khi tất cả các ô của hàng trên cùng (các ô từ 35 đến 41) đã được lấp đầy. Do đó ta sẽ thực hiện:

Bước 1 $i = 35$

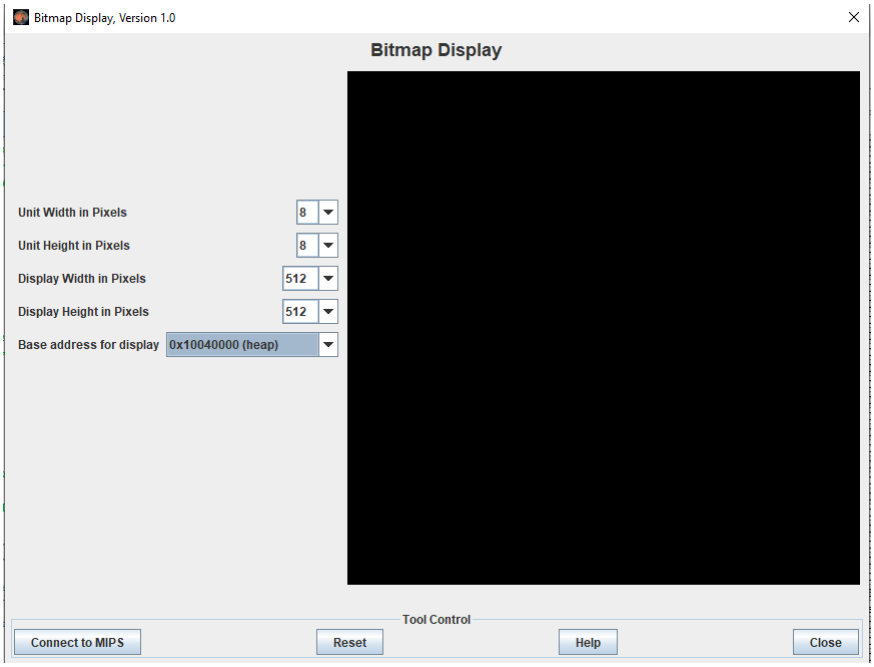
Bước 2 Nếu $i == 42$ thì ván đấu hòa. Kết thúc thuật toán.

Bước 3 Nếu $BoardArray\,[i] == 0$ thì ván đấu chưa kết thúc. Kết thúc thuật toán.

Bước 4 $i = i + 1$. Quay lại bước 2.

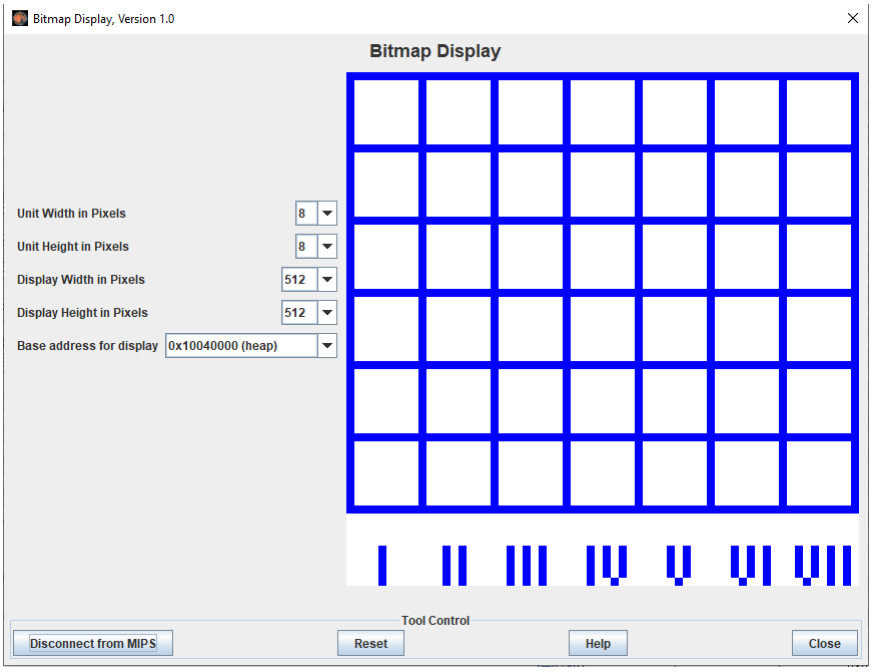
# 5 Kết quả

Đầu tiên, ta điều chỉnh các thông số của Bitmap Display như đã đề cập. Ấn vào nút *Connect to MIPS*, và bắt đầu chạy chương trình.



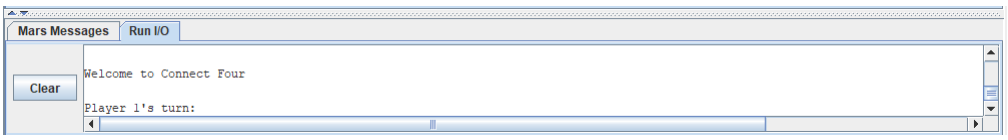

Hình 1: Điều chỉnh thông số của Bitmap Display

Ngay sau khi bắt đầu chương trình, Bitmap Display sẽ được load trở thành như hình bên dưới.



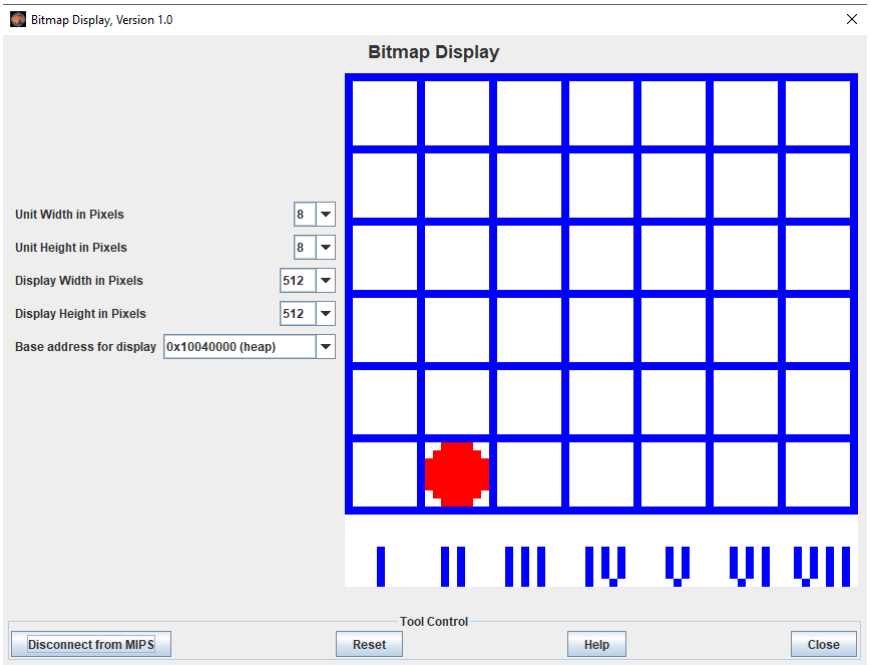

Hình 2: Bắt đầu

Ở dưới thanh *Run I/O* chính là nơi mà chúng ta sẽ chơi trò chơi, tức là 2 người chơi sẽ lần lượt nhập vào cột mà họ muốn thực hiện nước đi của mình.



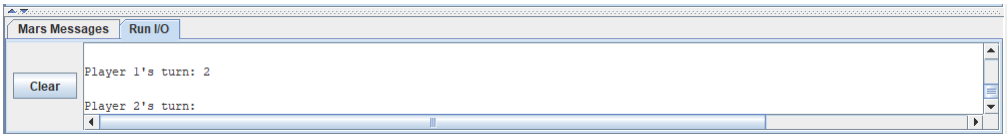

Hình 3: Lượt chơi

Giả sử người chơi 1 nhập vào 2, Bitmap Display sẽ hiển thị như hình vẽ. Đúng với một vòng tròn ở cột thứ 2.

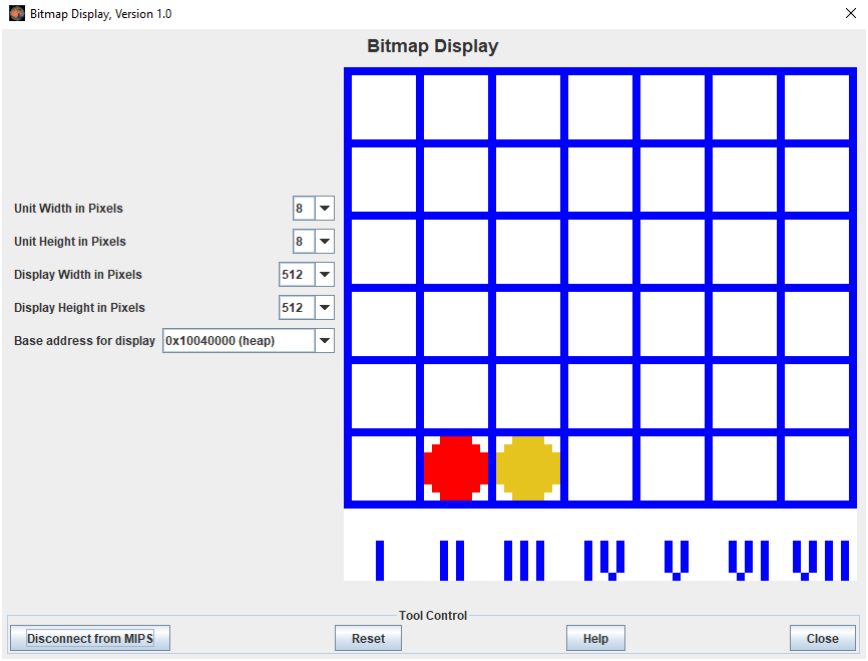



Hình 4: Hiển thị lượt chơi

Người chơi thứ 2 sẽ tiếp tục trò chơi. Thao tác giống người chơi đầu tiên.

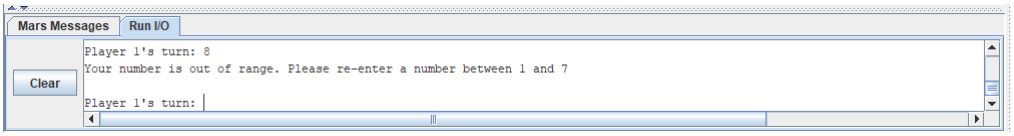



Hình 5: Lượt chơi khác



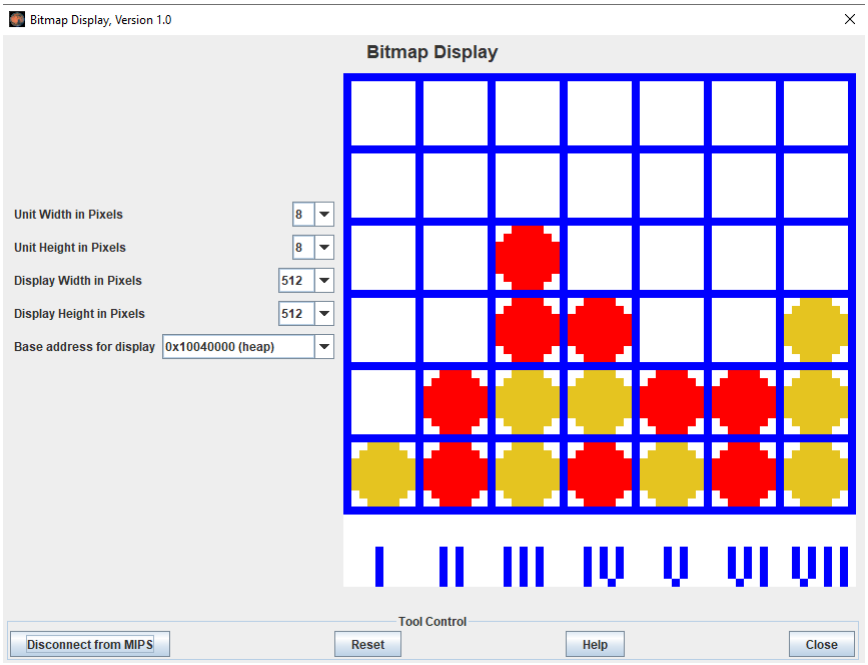

Hình 6: Hiển thị lượt chơi

Giả sử người chơi nhập vào một số không hợp lệ, chương trình sẽ hiện ra một thông báo như hình bên dưới.

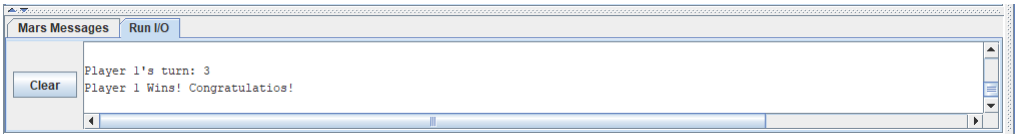



Hình 7: Số không hợp lệ

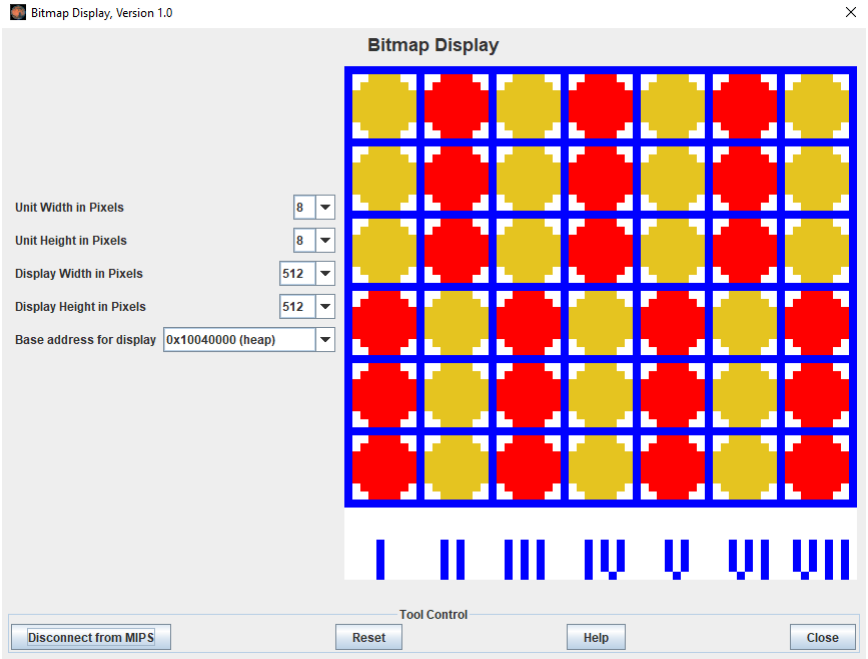
Hai người chơi sẽ tiếp tục cho đến khi nào có người thắng hoặc ván đấu hòa. Chương trình dừng lại.



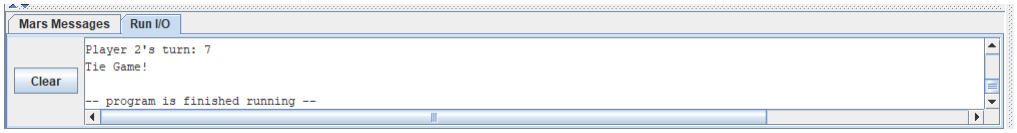

Hình 8: Người chơi 1 thắng




Hình 9: Người chơi 1 thắng




Hình 10: Ván đấu hòa




Hình 11: Ván đấu hòa

Như vậy, ta đã hoàn thành hiện thực trò chơi Connect Four với chế độ Multiplayer Game.

# Tài liệu tham khảo

[1] David A Patterson và John L.Hennessy. *Computer Organization and Design: The Hardware/Software Interface, Fifth Edition*. Morgan Kaufmann Publishers, 2017.

[2] Phạm Quốc Cường. *Kiến trúc máy tính*. Nhà xuất bản Đại học Quốc gia TPHCM.