How do we represent a complex problem?

- Outline what needs to be done (e.g., diagram of program steps)
  - Determine C++ variables to use
  - Determine C++ statements to use

The world of a video game

Two-dimensional grid containing:
- Players
- Walls
- Open spaces

Each player has:
- Location
- Facing-direction

The world of a video game (continued)

For each character, request an action

Possible actions
- Move forward
- Change direction
  - Player 1 move forward
  - Character 2 turn to face down

Progress of game

Representing the world

First approach:
- List locations of players and walls
  
  \[
  \text{int players[maxP][2];}
  \]
  \[
  \text{int walls[maxW][2];}
  \]
  \[
  \text{players[0][0]=2;}
  \]
  \[
  \text{players[0][1]=1;}
  \]
Representing the world

First approach:
- List locations of players and walls

```plaintext
players[0][0]=2;
players[0][1]=1;
walls[0][0]=2
walls[0][1]=3
```

 Players
1: row 3, col 2
2: row 2, col 7
3: row 6, col 5

Walls
1: row 3, col 4
2: row 3, col 5
3: row 5, col 7
4: row 7, col 8

Representing the world

Second approach:
- Record the contents of each location

```plaintext
char world[8][8];
world[0][0]='O';
world[0][1]='O';
world[1][6]='A';
world[2][1]='B';
world[2][3]='W';
world[2][4]='W';
(players labeled as A, B, C)
```

Moving through the world

To move player 1 forward by 1 box
- Find player current location
- Check if target location is empty
- Update player location

Find player 1 location

Approach 1:
- Look up `players[0][0]` and `players[0][1]`

Approach 2:
- Loop through each element of world, look for 'B'

**Approach 1 is faster!**

Check if target location empty

- Approach 1:
  - Look at locations of all players and walls – confirm none are xNew, yNew

- Approach 2:
  - Check `world[xNew][yNew] == 'O'`

**Approach 2 is faster!**

Abstraction

- Function – a set of actions called by one word
- Class – a set of data held in one word

Information hiding
- So long as action/data unit acts correctly, we don’t need to know the details
- Hiding details can prevent accidents in programming (e.g., overdrawn account)