CommandCenter Class

Introduction How to use

CommandCall(String command) is used to call a command. Its argument has two types: 1. a series of inputs and 2. the name of an action, as shown in the following examples for calling a fire ball.

```
fakename.commandCall(“2 3 6 _ A”);
or
fakename.commandCall(“STAND_D_DF_FA”);
```

For the first-type argument, each input must be split with space, and they will be executed from left to right. Numbers (2 3 6 in the above example) indicate movement directions as shown in the table below.

7( )	8( )	9( )
4( )	5(・)	6( )
1( )	2( )	3( )

(5 indicates the input for no pressing of any key)

Moreover, the symbol "_" is used to connect simultaneously pressed inputs. Some skills like throw require us to press direction and attack keys at the same time. In the above example, the direction “6” and attack “A” will be pressed at the same time.

If you use CommandCenter, as in the following example, you can always assume that your character always faces to the right-hand side, so that you can refer to above table. This is because even though your character's facing direction changes, the system will automatically reverse your command directions.

There is a point that has to be cautioned. While a command is being input, the system will not accept the input of a subsequent command. For this you can use the getSkillflag command. If a command is being input the flag will return 1, otherwise 0.

A sample AI ~~as~~

```
import structs.FrameData;
import structs.GameData;
import structs.Key;
import gameInterface.AIInterface;
import commandcenter.CommandCenter;

public class CCSample implements AIInterface {
    Key inputKey;
    boolean playerNumber;
    FrameData frameData;
    CommandCenter cc;

@Override
public int initialize(GameData gameData, boolean playerNumber) {
    this.playerNumber = playerNumber;
    this.inputKey = new Key();
    cc = new CommandCenter();
    frameData = new FrameData();
    return 0;
}

@Override
public void getInformation(FrameData frameData) {
    this.frameData = frameData;
    cc.setFrameData(this.frameData, playerNumber);
}

@Override
public void processing() {
    if(!frameData.emptyFlag && frameData.getRemainingTime() > 0){
        if(cc.getskillFlag()){
            inputKey = cc.getSkillKey();
        }else{
          inputKey.empty();
          cc.skillCancel();
          if(cc.getDistanceX() < 100){
              cc.commandCall("THROW_A");
          }else if(cc.getDistanceX() > 300){
              cc.commandCall("2 3 6 _ A");
          }
        }
    }
}
```

```
@Override
public Key input() {
	return inputKey;
}
}
```

Type	Field and Description
private boolean	skillflag Sign ~~of~~ a skill is being input.

Type	Method and Description
public Boolean	getSkillflag() If a command is being input, then returns true, otherwise false.
public void	setFrameData(FrameData framedata, Boolean playerNumber) Sets the frame data.
public void	commandCall(String command) Uses “String command” to create SkillData, an array containing the input sequence.
public void	skillCancel() Empties skillData and sets skillFlag to false.
public Key	getSkillKey() Gets the current Key data.
public int	getMyHP() Gets the hit point of self.
public int	getMyEnergy() Gets the energy value of self.
public int	getEnemyHP() Gets the hit point of the opponent.
public int	getEnemyEnergy() Get the energy value of the opponent.
public int	getMyX() Gets self’s x-coordinate.
public int	getMyY() Gets self’s y-coordinate.
public int	getEnemyX() Gets the opponent’s x-coordinate.

public int	getEnemyY() Gets the opponent's y-coordinate.
public int	getDistanceX() Gets the horizontal distance between the two characters.
public int	getDistanceY() Gets the vertical distance between the two characters.
public CharacterData	getMyCharacter() Gets the character data of self.
public CharacterData	getEnemyCharacter() Gets the character data ~~of~~ opponent.