**Aviso: Esta es una versión muy simplificada del código. Hay numerosos comentarios explicando su funcionamiento, y muchos métodos colocados para resumir la lógica de manera que no haga falta entrar en todo el código en sí para su comprensión. Se puede solicitar el archivo original con el código completo si se ve necesario**

**public void MoveMe(int mov)** //Método al que se llama para elegir de entre sus fichas cuál debe mover (mov es la tirada del dado)

{

Player jugAct = GameManager.getActPlayer(); //Pilla de sus jugadores el jugador actual

Bool[] cuales;

int numJugables = 0; //Primero debe de comprobar cuantas fichas puede mover (por barreras o por que estén en casa)

for(int i = 0; i < jugAct.misFichas.Length; i++)

{

if (jugAct.misFichas[i].canMove(mov)) //Método que comprueba si una ficha puede moverse

{

cuales[i] = true; //Si puede mover la ficha, guarda en el array que puede

numJugables++; moverla y suma 1 al número de fichas jugables

}

}

//Una vez ha comprobado todas las fichas, decide qué hacer

if(numJugables == 0) //Si no puede jugar ninguna ficha, pasa turno

{

GameManager.NextTurn();

}

else

{

if (numJugables == 1) //Si solo puede jugar una ficha, se ve forzado a moverla

{

for(int i = 0; i < cuales.Length; i++)

{

if (cuales[i]) jugAct.MoveFicha(i); //Busca entre las fichas cual es la que puede mover, y la mueve

}

}

else //Si no ha pasado lo uno ni lo otro, habrá que elegir la ficha a mover

{

int[] ValorFichs; //Lista que guarda el valor de mover cada ficha

for(int k = 0; k < jugAct.misFichas.Length; k++) //Método que llama a todas las fichas y comprueba el valor de moverlas

{ //Aquí también se hace un filtrado para no llamar a las fichas que no se pueden mover

ValorFichs [k] = **calculate**(k, mov); //Se llama al método calculate del programa para calcular el valor de mover la ficha

}

int posMax = GetGreatestResult(ValorFichs); //Método para pillar cuál de las posiciones de la lista es la mayor

jugAct.MoveFicha(posMax);

}

}

}

**int calculate(int ficha, int mov)** //Método que resta el valor de la posición en la que se está el valor de la posición a la que se podría mover

{

Player jugAct = GameManager.getActPlayer();

return (**valuePos**(jugAct.misFichas[ficha].pos + mov) –

**valuePos**(jugAct.misFichas[ficha].pos);

//el valor devuelto es el de la posición a la que se movería restándole el de la posición actual. Por lo tanto, si está en una posición buena, y se mueve a una mala, el valor sería negativo, y lo mismo al revés. (Ejemplo, está en una posición segura de valor 3, y de moverse acabaría en una posición insegura de valor -3. Por lo tanto la ecuación sería de -3 – 3 = -6. Por lo que sería un movimiento muy malo)

}

**int valuePos(int pos)** //Método que calcula el valor de una posición en el tablero

{

Player jugAct = GameManager.getActPlayer();

int value = 0; //Variable que guardará el valor de esta posición

if (CasillasdeMeta(pos)) //Si la posición es la de una casilla del trecho final único del jugador (donde no puede ser comido ni atacado)

{

//isRisky() es un método que elige si hacer una jugada arriesgada o segura, según las configuraciones elegidas en el menú principal

if(isRisky()) value += 2;

else value += 4;

if (pos == CasillaFinal) //Si te mueves a la casilla final, y puedes avanzar 10 casillas

{

value += 2;

//Se llama a un método que comprueba el valor de todas las fichas posibles si avanzasen 10 casillas, y pilla el mejor. Si todas las opciones de mover 10 casillas fueran malas, el valor sería negativo, y tal vez compensaría mover otra ficha en su lugar

value += CheckExtraMovement(10) \* 2;

}

}

else

{

if (Tablero.Casillas[pos].safe) //Si la casilla es una casilla segura, habrá que compensar colocarse ahí

{

if (isRisky()) value += 2;

else value += 3;

if(CreoBarrera()) //Si que esa ficha esté ahí crea una barrera, eso es algo positivo

{

value += 3;

}

//Si estoy en mi posición de inicio con una barrera y aún tengo fichas en casa, eso significa que estoy potencialmente desperdiciando cincos para sacar fichas, por lo que tal vez me interesaría mover esa barrera para así poder sacar esa ficha

if (pos == jugAct.initPos && jugAct.TengoFichasSinSacar())

{

if (isRisky()) value -= 1;

else value -= 4;

}

//Luego debe comprobar si está en la casilla de inicio de otro jugador, si ya hay otra ficha ahí, y si le quedan fichas por sacar. En ese caso, si sacase un 5 podría comerse su ficha, por lo que tal vez le interese salir de ahí

If(CasillaIniOtroJug(pos) && Tablero.Casillas[pos].numFichas == 2 && FichasPorSacarOtroJugador())

{

if (isRisky()) value -= 2;

else value -= 4;

}

}

else

{

if (ComoFicha()) //Si me como una ficha rival

{

if (isRisky()) value += 10;

else value += 3;

value += CheckExtraMovement(20) //Cuando comes una ficha rival avanzas 20 casillas, por lo que también habría que ver si compensa avanzar 20 casillas en este momento

}

//Si se coloca en una posición no segura delante de fichas rivales, estas le podrían comer, por lo que hay que evitarlo con un método que reste según el número de fichas que haya detrás suya y lo lejos que estén

if (isRisky()) value -= TengoFichasEnemigasAtras(pos);

else value -= TengoFichasEnemigasAtras(pos) \* 2;

// Y lo mismo, si tiene fichas delante suya, tal vez podría comérselas en su siguiente turno, y le compense colocarse ahí

if (isRisky()) value += TengoFichasEnemigasDelante(pos) \* 2;

else value += TengoFichasEnemigasDelante(pos);

}

//Finalmente, una vez se han hecho todos los cálculos, se devuelve el valor que da como resultado

return value;

}