Projecte: Escacs

Metodologia de la Programació

Curs 2021 - 2022

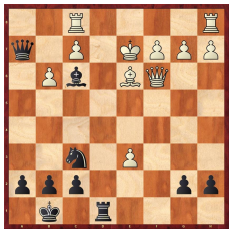
Objectius de la sessió

1. Presentació i organització del projecte
2. Explicar les tasques a realitzar a la primera versió del projecte

Organització del projecte

- **Grups de 2 persones:** hem obert a Caronte una inscripció a grups perquè pugueu indicar amb qui fareu el projecte.
  - Apunteu-vos a un dels subgrups corresponents al vostre grup de classe.
- El projecte és part del **treball autònom** de l'assignatura. Durant les hores de classe dedicarem algunes sessions a seguiment i avaluació del treball.

Objectiu del projecte



Disseny i desenvolupament d'una versió completament funcional del joc del **escacs**, posant en pràctica els conceptes que anem explicant a les sessions de classe.

El vostre programa final haurà de permetre:

- **Inicialitzar una partida** des de zero o a partir d'un **estat inicial** guardat en un fitxer de text.
- **Jugar la partida** a partir de l'estat inicial. A cada **torn** d'un jugador:
  - **Seleccionar una peça** per moure.
  - Determinar tots els **moviments vàlids** que pot fer la peça.
  - **Moure** la **peça** a una nova posició dins dels moviments vàlids.
  - **Actualitzar** l'estat del **tauler** tenint en compte si es mata alguna peça del contrari.
- **Guardar** tots els **moviments** que es fan durant la partida en un fitxer.
- **Reproduir** una **partida** prèviament jugada executant els moviments guardats en un fitxer.

Podreu afegir **funcionalitats extres**, si voleu, que comptaran positivament com a **punts addicionals** a la **nota final** del projecte.

Planificació del projecte

El projecte el desenvoluparem en dues fases, que es correspondran amb el lliurament parcial i el lliurament final del projecte.

Primera versió del projecte:

- Inicialitzar el tauler del joc a partir de la informació guardada a un fitxer de text.
- Determinar els moviments vàlids de qualsevol peça del tauler.
- Moure una peça, comprovant que el moviment és vàlid.
- Mostrar l'estat actual del tauler.
- En aquesta primer versió treballarem sense visualització gràfica. Tindreu un test d'autoavaluació a Caronte per poder validar el correcte funcionament de les diferents funcionalitats.

Segona versió del projecte:

- Implementar la part gràfica del joc i la interacció del jugador amb el tauler durant el seu torn.
- Implementar el desenvolupament complet d'una partida a partir d'un estat inicial, alternant els torns dels jugadors fins al final de la partida.
- Guardar en un fitxer els moviments que es fan durant el desenvolupament de la partida.
- Reproduir una partida prèviament jugada executant els moviments guardats en un fitxer.

Primera versió del projecte

Inicialitzar una partida

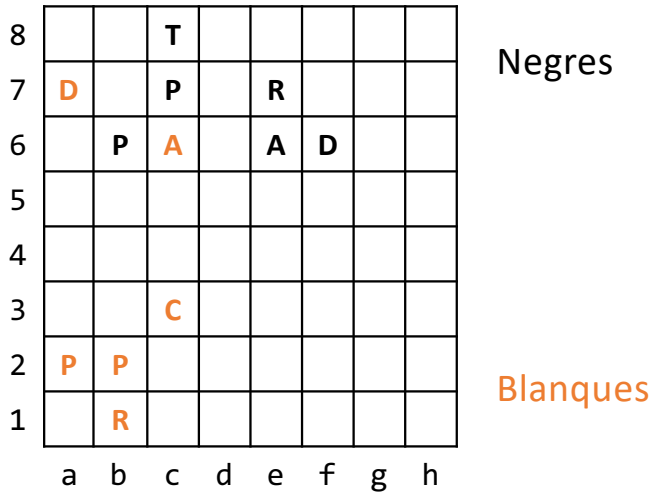
S'haurà de poder inicialitzar l'estat inicial de la partida a partir de la informació guardada en un fitxer. El fitxer contindrà la informació de la posició inicial de cada peça seguint el format que es mostra en aquest exemple:

```
0. Rb1
0. Da7
0. Ac6
0. Cc3
0. Pa2
0. Pb2
1. Re7
1. Df6
1. Tc8
1. Ae6
1. Pb6
1. Pc7
```

Codificació del jugador:
- 0: blanques
- 1: negres

Tipus de la peça:
- R: Rei
- D: Dama
- T: Torre
- A: Alfil
- C: Cavall
- P: Peó

Posició dins del tauler:



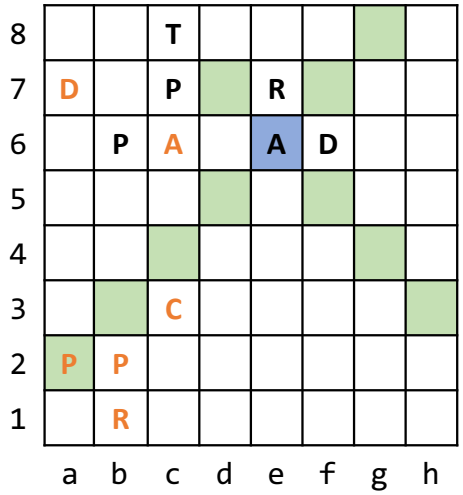
Primera versió del projecte

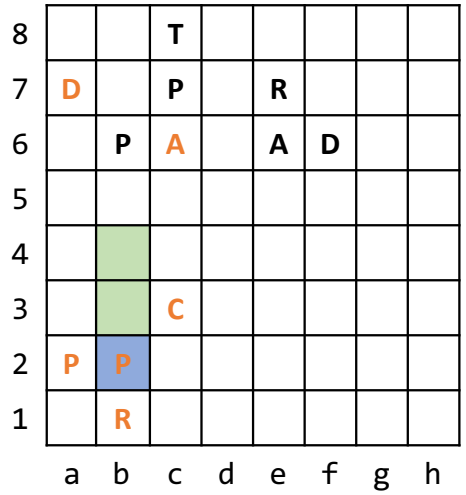
Tipus de la peça:
- R: Rei
- D: Dama
- T: Torre
- A: Alfil
- C: Cavall
- P: Peó

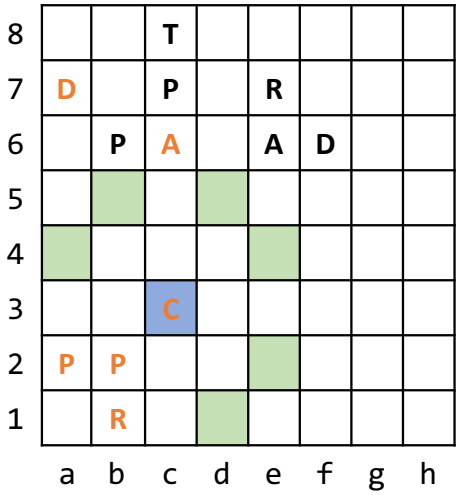
Determinar els moviments vàlids d'una peça

Donada una posició del tauler, s'haurà de poder recuperar el conjunt de posicions a on es pot moure la peça que ocupa aquella posició tenint en compte:

- Només es poden fer els moviments permesos en funció del tipus de peça.
- Una peça no es pot moure a una posició ocupada per una altra peça del mateix jugador, però sí que es pot moure a una posició ocupada per una peça del jugador contrari.
- Una peça no pot avançar més enllà d'una posició ocupada per una altra peça, ja sigui del mateix jugador o del jugador contrari.







Tipus de la peça:
- R: Rei
- D: Dama
- T: Torre
- A: Alfil
- C: Cavall
- P: Peó

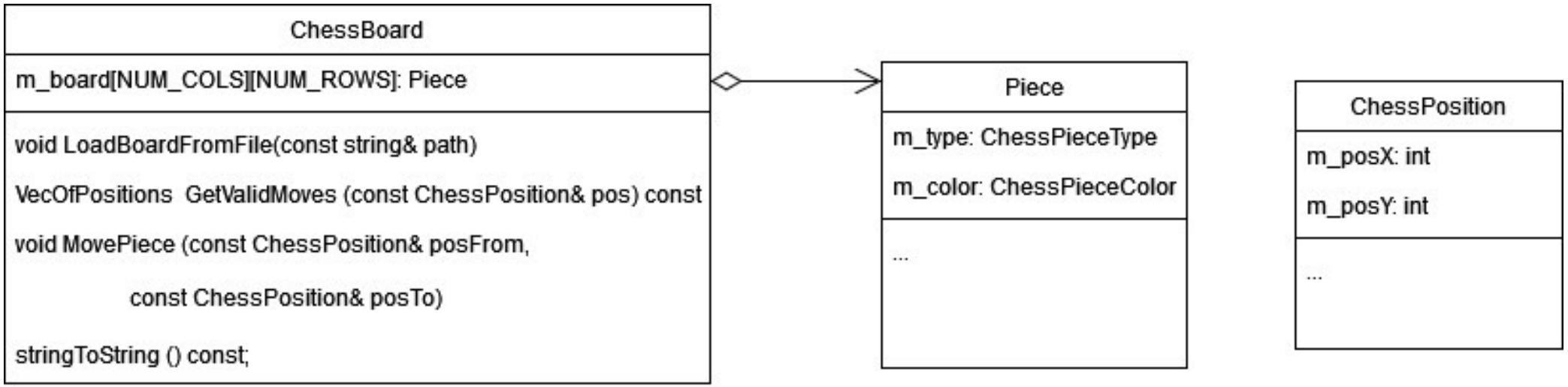
Primera versió del projecte

Recordem: moviments de les peces

- **Rei**: pot avançar una sola posició en qualsevol de les 8 direccions possibles.
- **Dama**: pot avançar tantes posicions com vulgui en qualsevol de les 8 direccions possibles.
- **Torre**: pot avançar tantes posicions com vulgui en qualsevol de les direccions horitzontal o vertical.
- **Alfil**: pot avançar tantes posicions com vulgui en qualsevol de les direccions diagonals.
- **Cavall**: avança en forma de 'L', dues posicions en horitzontal o vertical i una posició perpendicular al primer moviment.
- **Peó**: si està a la seva posició inicial al tauler, pot avançar una o dues posicions endavant. Si està a qualsevol altra posició, només pot avançar una posició endavant. Per matar, pot avançar una posició en diagonal cap a qualsevol dels dos costats.
- **No considerarem** altres tipus de **moviments** o accions **especials** a la versió bàsica mínima del joc (ho podeu incloure si voleu com a funcionalitats extres):
  - Enroc entre el rei i la torre.
  - Canvi del peó per una altra peça si arriba al final del tauler.

Primera versió del projecte

Estructura de classes



Primera versió del projecte

Classe `Piece`

- Guarda la informació d'una peça del tauler de joc
- Té atributs per guardar el color (de quin jugador és) i el tipus de la peça.
- Afegiu els mètodes que facin falta, bàsicament constructor, getters i setters i també algun mètode per convertir la peça en un string (serà necessari per mostrar la informació del tauler per pantalla).
- Per codificar el tipus i el color de les peces us donem ja declarats aquests dos tipus `enumerate`:

Tipus de la peça:
- R: Rei
- D: Dama
- T: Torre
- A: Alfil
- C: Cavall
- P: Peó

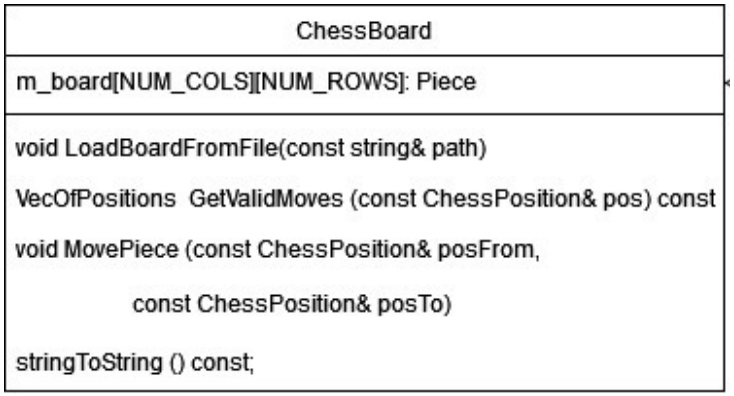
```
typedef enum {
    CPT_King,
    CPT_Queen,
    CPT_Rook,
    CPT_Bishop,
    CPT_Knight,
    CPT_Pawn,
    CPT_EMPTY
} ChessPieceType;
```

```
typedef enum {
    CPC_Black,
    CPC_White,
    CPC_NONE
} ChessPieceColor;
```

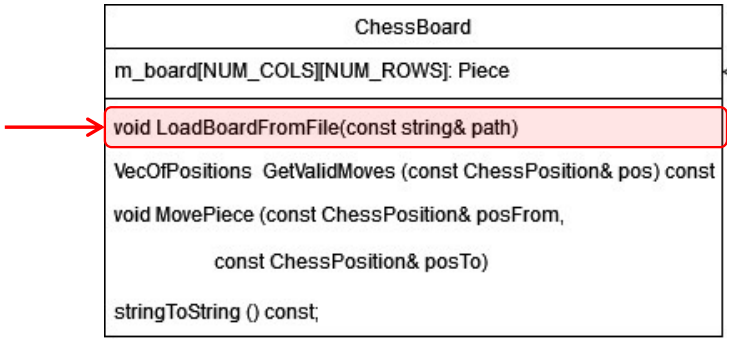
Primera versió del projecte

Classe `ChessBoard`

- Guarda la informació del tauler del joc
- Té un atribut per guardar una matriu on cada posició correspon a una de les caselles del joc i conté la informació de la peça que hi ha en aquella posició.
- A les posicions on no hi ha cap peça es pot guardar una peça de tipus `CPT_EMPTY`
- Haureu d'implementar els mètodes públics de la classe `ChessBoard`. Per implementar-los haureu d'implementar altres mètodes privats auxiliars que us ajudin a estructurar el codi seguint els principis de la descomposició modular.



Primera versió del projecte



Classe ChessBoard: mètodes

```
void LoadBoardFromFile(const string& path)
```

- Inicialitza el tauler de joc a partir de la informació d'un fitxer de text en el format que s'ha explicat abans

```
0. Rb1
0. Da7
0. Ac6
0. Cc3
0. Pa2
0. Pb2
1. Re7
1. Df6
1. Tc8
1. Ae6
1. Pb6
1. Pc7
```

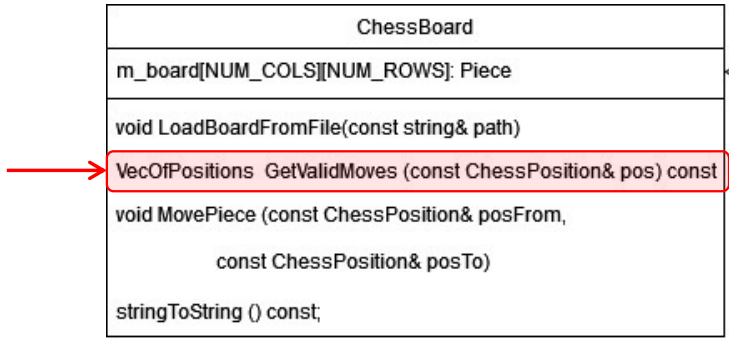
Codificació del jugador:
- 0: blanques
- 1: negres

Tipus de la peça:
- R: Rei
- D: Dama
- T: Torre
- A: Alfil
- C: Cavall
- P: Peó

Posició dins del tauler:

	a	b	c	d	e	f	g	h	
8			T						
7	D		P		R				Negres
6		P	A		A	D			
5									
4									
3			C						
2	P	P							Blanques
1		R							

Primera versió del projecte

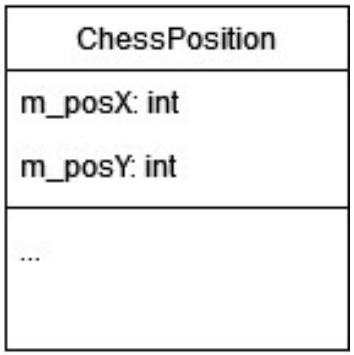


Classe ChessBoard: mètodes

```
VecOfPositions  GetValidMoves (const ChessPosition& pos) const
```

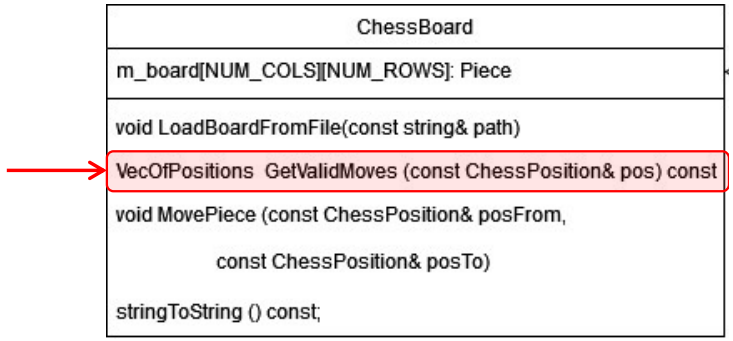
- Calcula tots els moviments vàlids que pot fer la peça que ocupa la posició que es passa com a paràmetre.
- Els moviments vàlids es retornen com el conjunt de posicions a les es pot moure la peça.
- Per guardar cadascuna de les posicions es pot fer servir la **classe ChessPosition.**

Classe ChessPosition



- Guarda la fila i la columna d'una posició del tauler
- Afegiu els mètodes que facin falta, bàsicament constructor, getters i setters i també algun mètode per convertir la posició en un string o inicialitzar-la a partir d'un string.
- Per poder executar el test que us donarem a Caronte:
  - Cal que tingui un constructor que rebi com a paràmetre un string en el format que hem indicat abans quan hem explicat la inicialització de la partida.
  - Cal que tingui definida la sobrecàrrega de l'operador == per poder comparar si dues posicions són iguals o no.

Primera versió del projecte



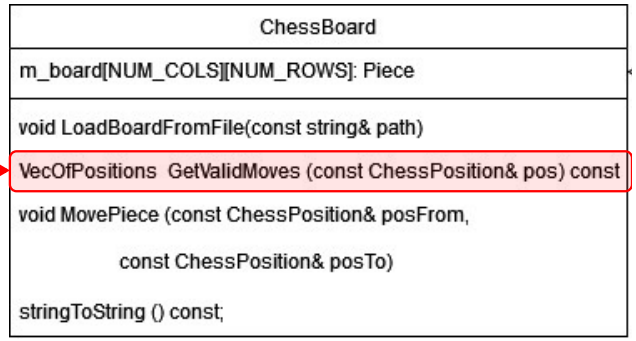
Classe ChessBoard: mètodes

```
VecOfPositions  GetValidMoves (const ChessPosition& pos) const
```

- El mètode retorna totes les posicions vàlides en un vector (utilitzant la classe de la llibreria estàndard `vector` que expliquem més endavant).
- Al codi inicial que us donarem trobareu aquesta declaració del tipus `VecOfPositions` per guardar aquest vector de posicions:

```
typedef vector<ChessPosition> VecOfPositions;
```

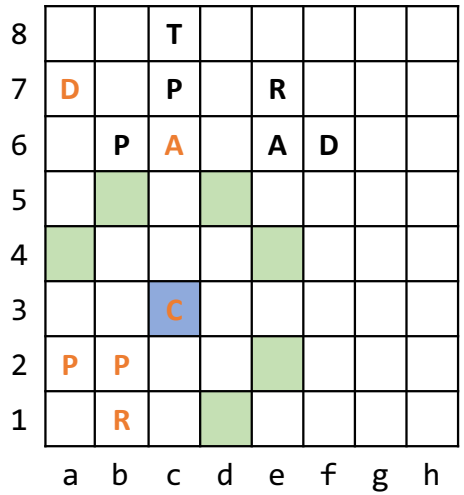
Primera versió del projecte



Classe ChessBoard: mètodes

```
VecOfPositions  GetValidMoves (const ChessPosition& pos) const
```

Exemple

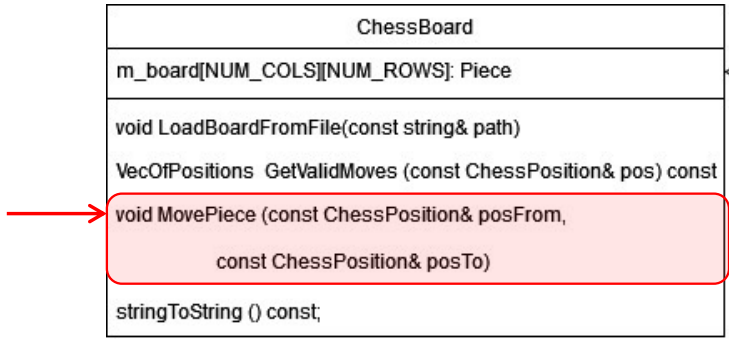



```
GetValidMoves(ChessPosition('c3'))
```

↓

```
[ChessPosition('a4'),
 ChessPosition('b5'),
 ChessPosition('d5'),
 ChessPosition('e4'),
 ChessPosition('e2'),
 ChessPosition('d1')]
```

Primera versió del projecte

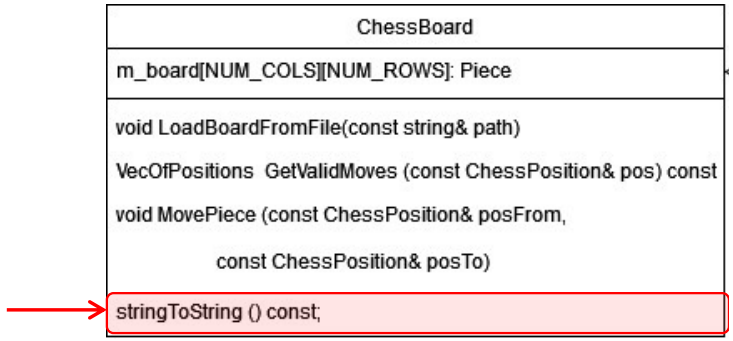


Classe ChessBoard: mètodes

```
bool MovePiece (const ChessPosition& posFrom, const ChessPosition& posTo)
```

- Mou la peça que ocupa la posició del paràmetre `posFrom` a la posició del paràmetre `posTo`.
- Ha de comprovar que la posició destí estigui dins dels moviments vàlids de la peça.
- Si no ho està no fa el moviment i retorna `false`.

Primera versió del projecte



Classe ChessBoard: mètodes

```
string ToString() const
```

- Genera un string amb l'estat actual del tauler de joc en el format que especifiquem a continuació

```
8 __ __ wR __ __ __ __ wR
7 bQ __ wP __ wK wP wP wP
6 __ wP bB __ wB wQ __ __
5 __ __ __ __ __ __ __ __
4 __ __ __ __ __ __ __ __
3 __ __ bH __ wP __ __ __
2 bP bP bP __ __ __ bP bP
1 __ bK __ bR __ __ __ __
  a  b  c  d  e  f  g  h
```

Codificació del jugador:
- 0: blanques (w)
- 1: negres (b)

Tipus de la peça:
- R: Rei
- D: Dama
- T: Torre
- A: Alfil
- C: Cavall
- P: Peó

Organització del projecte

MARÇ 2022						
dl	dm	dx	dj	dv	ds	dg
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	**23**	24	25	26	27
28	29	30	31			

ABRIL 2022						
dl	dm	dx	dj	dv	ds	dg
				1	2	3
4	5	6	7	**8**	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	**29**	30	

MAIG 2022						
dl	dm	dx	dj	dv	ds	dg
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

LLEGENDA		
D	Inici del projecte	23/3/22
D	Seguiment del projecte	8/4/22
D	Entrega primera versió del projecte	29/4/22

- La propera sessió de seguiment del projecte serà el **divendres 8 d'abril**.
  - Revisió del que heu avançat en el projecte i de com plantegeu el disseny de les classes `Piece` i `ChessPosition` i la implementació dels mètodes de la classe `ChessBoard`.
  - Comptarà un **20% de la nota del lliurament parcial**.
- El lliurament parcial del projecte amb la implementació de la primera versió serà el divendres **29 d'abril**. A la setmana següent farem una avaluació online individual del lliurament de cada grup.
- A Caronte tindreu un **test d'autoavaluació** dels mètodes públics de la classe `ChessBoard` que heu d'implementar.

Propers passos: alguns consells

1. Penseu quins mètodes us poden fer falta a la classe `Piece`. Feu una primera implementació de la classe.
2. Penseu quins mètodes us poden fer falta a la classe `ChessPosition`. Feu una primera implementació de la classe.
3. Feu la implementació del mètode `LoadBoardFromFile` de la classe `ChessBoard` per poder inicialitzar el tauler de fitxer. Si us fa falta afegir o modificar algun mètode a les classes `Piece` i `ChessPosition`, adapteu la implementació d'aquestes classes.
4. Penseu com podeu descomposar el codi del mètode `GetValidMoves` amb funcions més petites i simples.
   - Com podem organitzar el codi del mètode per poder implementar fàcilment el codi per recuperar els moviments vàlids per cada tipus de peça?
   - Hi ha parts del codi que siguin comuns a diferents tipus de peces? Com podem organitzar el codi per evitar duplicar aquestes parts comunes del codi?
5. Comenceu a implementar el codi del mètode `GetValidMoves`, per cada tipus de peça.

Classe `vector`

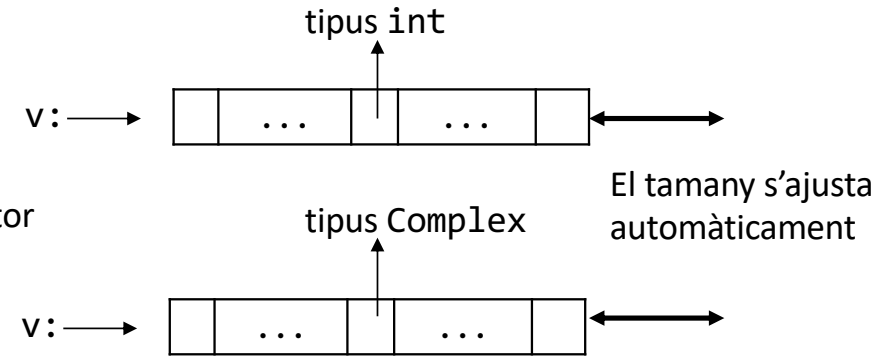
Classe `vector` https://en.cppreference.com/w/cpp/container/vector

- Permet gestionar arrays de longitud indefinida de qualsevol tipus
  - El tipus de l'array s'especifica en el moment de la declaració de l'array utilitzant *templates*.
- La mida de l'array s'ajusta automàticament al número d'elements que hi tenim guardats:
  - Si el número d'elements creix molt, automàticament es reserva més memòria i es fa una còpia de tots els elements al nou espai de memòria.

```
#include <vector>
using namespace std;

vector<int> v;
vector<Complex> v;
```

Tipus dels elements del vector

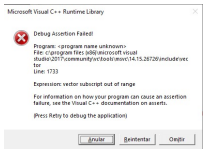


Classe `vector`

Classe `vector`: accés als elements

- L'accés es pot fer per índex (començant per 0) igual que als arrays.
- Només es pot accedir als índexs que corresponen a posicions vàlides (entre 0 i el nº d'elements actuals del vector).
- Inicialment, el nº d'elements del vector és 0. Si intentem accedir a qualsevol element tindrem un error d'accés
- Podem recuperar el nº d'elements actual cridant al mètode `size()`
- El nº d'elements actuals del vector canvia quan:
  - Afegim un element al vector (operacions `push_back`, `insert`)
  - Eliminem un element al vector (operacions `pop_back`, `erase`)
  - Redimensionem explícitament el tamany del vector (operació `resize`)

```
vector <int> v;
v[0] = 1;
```



v: ⟶

v.size() ⟶ 0

```
vector <int> v;
v.push_back(0);
v[0] = 10;
```

v: ⟶ | 10 |

v.size() ⟶ 1

```
vector <int> v;
v.resize(5);
for (int i = 0; i < v.size(); i++)
   v[i] = i;
```

v: ⟶ | 0 | 1 | 2 | 3 | 4 |

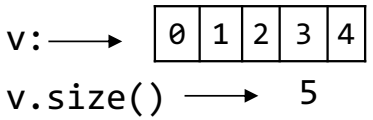
v.size() ⟶ 5

Classe `vector`

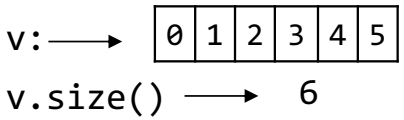
Classe `vector`: afegir/eliminar al final del vector

- **`push_back(element)`**: afegeix un element al final del vector, incrementant el nº d'elements actual.
- **`pop_back()`**: elimina l'últim element del vector, decrementant el nº d'elements actual

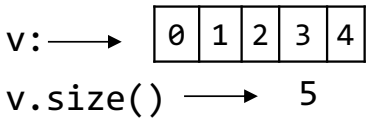
```
vector <int> v;
v.resize(5);
for (int i = 0; i < v.size(); i++)
   v[i] = i;
```




```
v.push_back(5);
```




```
v.pop_back();
```



Classe `vector`

Exemple

```
class Estudiant
{
private:
   string m_nom;
   string m_NIU;
   vector<string> m_assignatures;
};
```

```
void Estudiant::afegeixAssignatura(const string& assignatura)
{
   m_assignatures.push_back(assignatura);
}
```

```
void Estudiant::mostraAssignatures()
{
   for (int i = 0; i < m_assignatures.size(); i++)
      cout << m_assignatures[i] << endl;
}
```