

# 財務演算法期末





R10723041林大中

R10723050陳韻帆

R10723059胡祖望

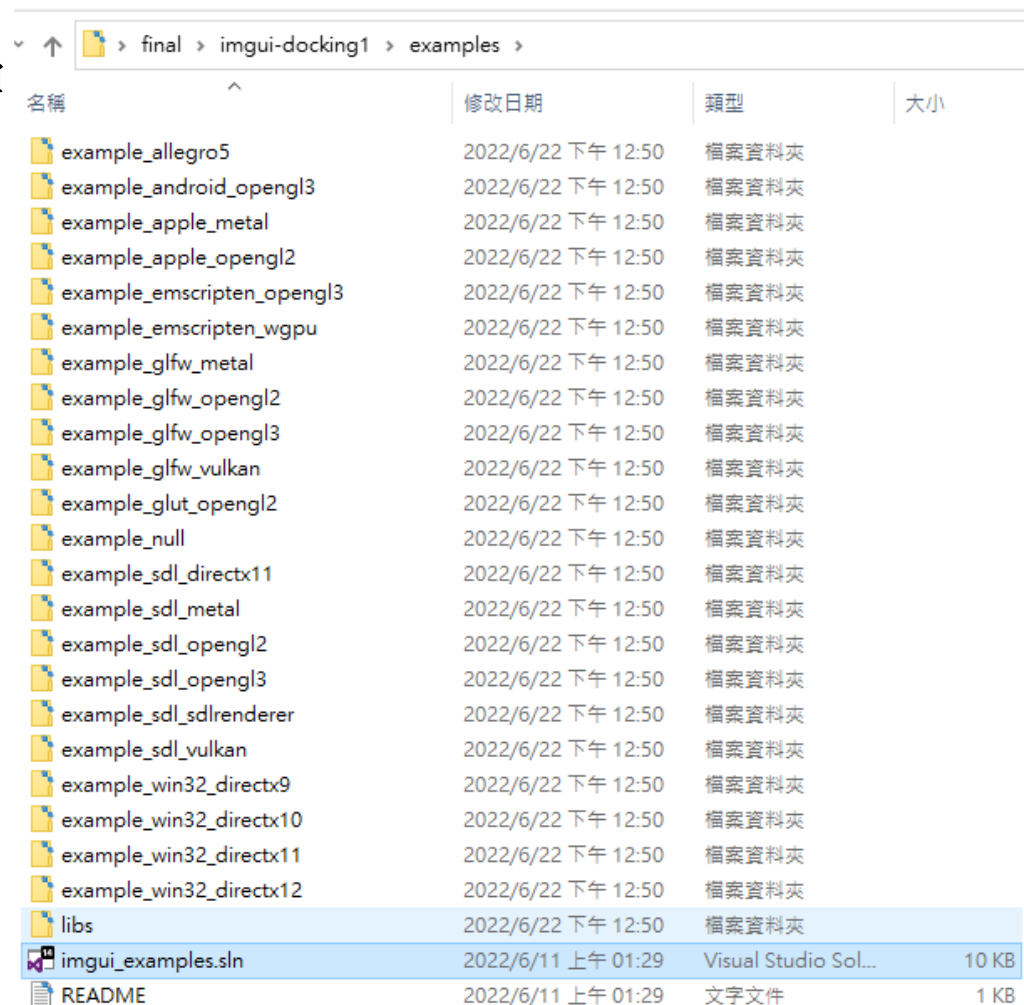
# 壓縮檔內容

- 資料夾是程式碼
- FINAL 是期末exe檔
- 還有此簡報

 imgui-docking1	2022/6/22 下午 12:50	檔案資料夾	
 final	2022/6/14 下午 02:24	應用程式	497 KB
 imgui	2022/6/22 下午 01:07	組態設定	1 KB
 簡報1	2022/6/14 下午 02:32	Microsoft Power...	1,037 KB

# 程式碼

- 打開程式碼請按路徑點，並按下圖片中  
Imgui\_examples.sln，  
開啟後執行example\_win32\_directx9



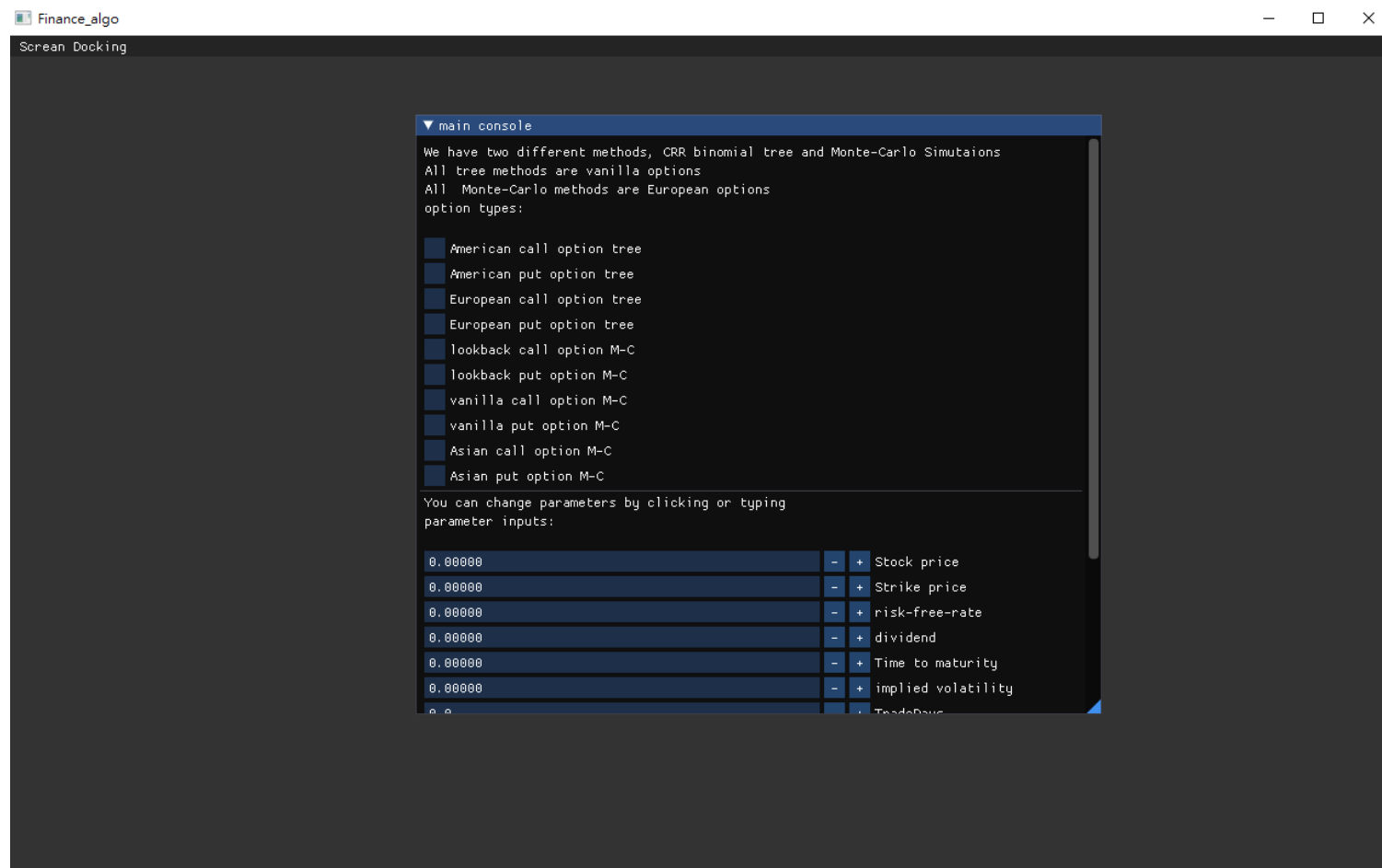
名稱	修改日期	類型	大小
example_allegro5	2022/6/22 下午 12:50	檔案資料夾	
example_android_opengl3	2022/6/22 下午 12:50	檔案資料夾	
example_apple_metal	2022/6/22 下午 12:50	檔案資料夾	
example_apple_opengl2	2022/6/22 下午 12:50	檔案資料夾	
example_emscripten_opengl3	2022/6/22 下午 12:50	檔案資料夾	
example_emscripten_wgpu	2022/6/22 下午 12:50	檔案資料夾	
example_glfw_metal	2022/6/22 下午 12:50	檔案資料夾	
example_glfw_opengl2	2022/6/22 下午 12:50	檔案資料夾	
example_glfw_opengl3	2022/6/22 下午 12:50	檔案資料夾	
example_glfw_vulkan	2022/6/22 下午 12:50	檔案資料夾	
example_glut_opengl2	2022/6/22 下午 12:50	檔案資料夾	
example_null	2022/6/22 下午 12:50	檔案資料夾	
example_sdl_directx11	2022/6/22 下午 12:50	檔案資料夾	
example_sdl_metal	2022/6/22 下午 12:50	檔案資料夾	
example_sdl_opengl2	2022/6/22 下午 12:50	檔案資料夾	
example_sdl_opengl3	2022/6/22 下午 12:50	檔案資料夾	
example_sdl_sdlrenderer	2022/6/22 下午 12:50	檔案資料夾	
example_sdl_vulkan	2022/6/22 下午 12:50	檔案資料夾	
example_win32_directx9	2022/6/22 下午 12:50	檔案資料夾	
example_win32_directx10	2022/6/22 下午 12:50	檔案資料夾	
example_win32_directx11	2022/6/22 下午 12:50	檔案資料夾	
example_win32_directx12	2022/6/22 下午 12:50	檔案資料夾	
libs	2022/6/22 下午 12:50	檔案資料夾	
imgui_examples.sln	2022/6/11 上午 01:29	Visual Studio Sol...	10 KB
README	2022/6/11 上午 01:29	文字文件	1 KB

# 功能介紹

- 功能有以下幾個
  1. 視窗介面移動與貼齊
  2. 計算選擇權
  3. 開新視窗

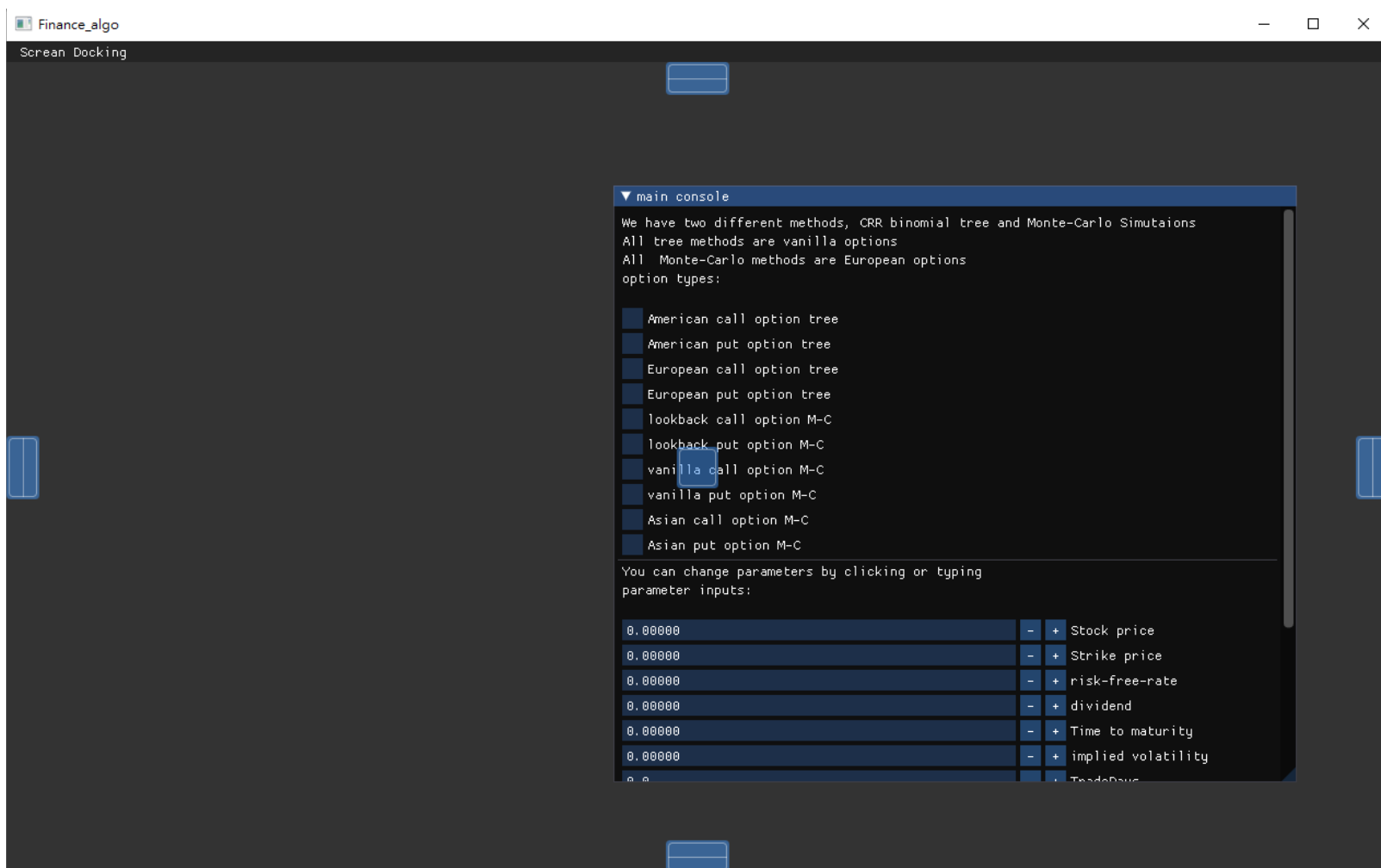
# 介面介紹

## 1.視窗縮放



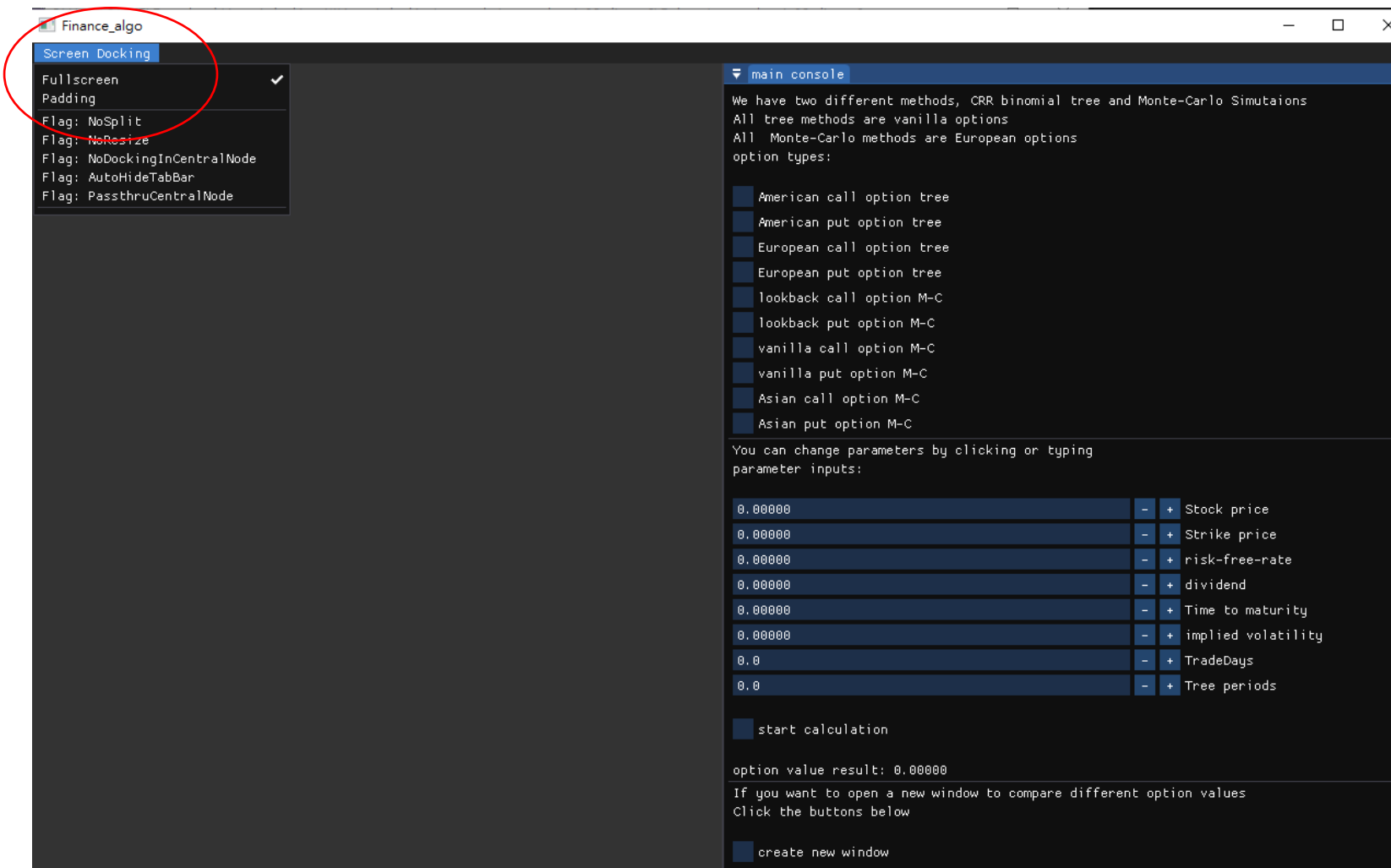
# 介面介紹

## 2.視窗貼齊，可貼齊上下左右與全螢幕



# 介面介紹

## 3.其餘介面的 功能可以在左上角的screen docking 中選取



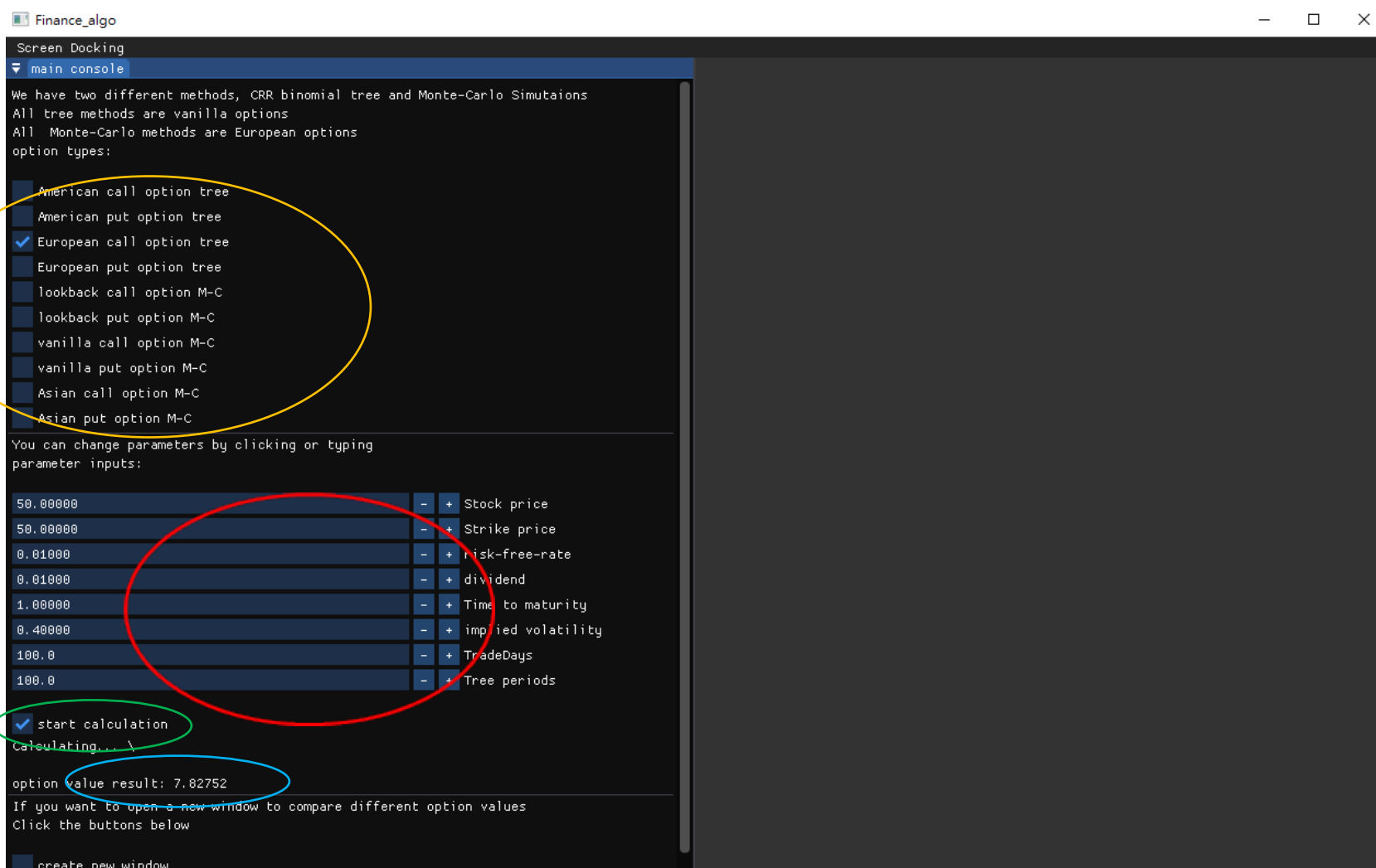
# 計算選擇權

選想求出的選擇  
權類型(選一個)

參數輸入

按開始

結果



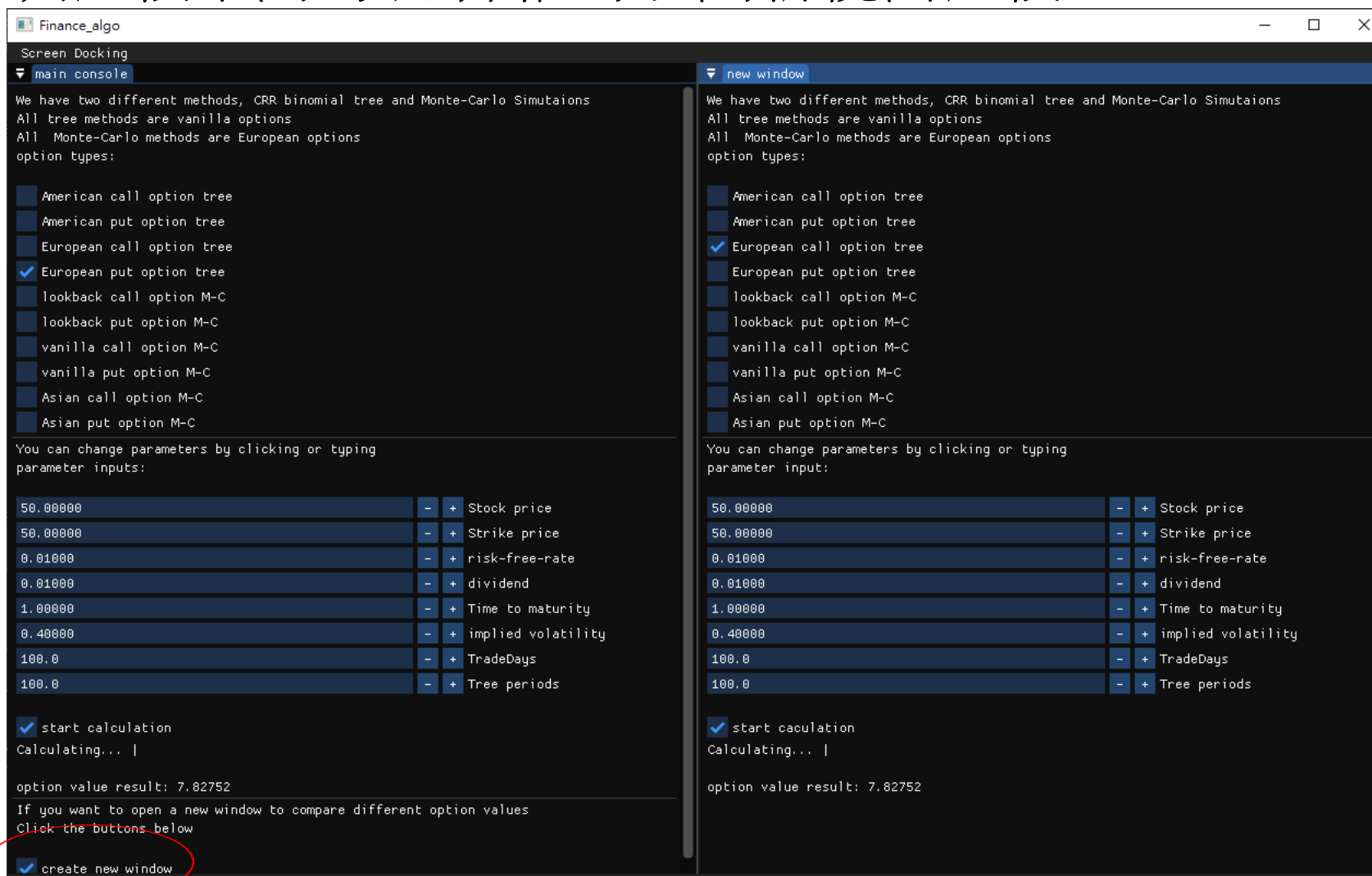


# 計算選擇權

- 選擇權一次選一個
- TREE 的計算很快
- Monte-Carlo 的計算比較慢，介面會比較卡

# 開新視窗

- 如果想要一次比較不同的 選擇權可以開新視窗比較



開新式窗

# 程式碼說明

# Simulator類

```
class Simulator {  
private:  
    array3d price;  
    vector<double> reductionRV;  
    double S, q, r, T, sigma;  
    int Rep, Sim, TradeDays;  
  
    // Normal RV  
    double getNormal();  
  
    // calculate price and put in array3d price  
    void calc();  
};
```

```
public:  
    // Constructor and destructor  
    Simulator(double S, double r, double q, double T, double sigma, int Rep, int Sim, int TradeDays);  
    virtual ~Simulator();  
  
    // getter  
    const array3d &getPrice() const;  
  
    double getS() const;  
  
    double getQ() const;  
  
    double getT() const;  
  
    double getSigma() const;  
  
    int getRep() const;  
  
    int getSim() const;  
  
    double getR() const;  
  
    int getTradeDays() const;  
};
```

- 用於生成path-dependent simulation by BSM，可藉由參數傳入不同的Payoff裡面可以作為中間媒介生成選擇權價格

# Payoff類

```
class Payoff {
public:
    // constructor and destructor
    Payoff();
    virtual ~Payoff();
    // store the option values for every simulation
    vector<double> optionValues;

    // override method
    virtual void calc() = 0;
    virtual void deletePrice() = 0;

    // get the simulation statistic summary result
    vector<double>& getValues();
    double getMean();
    double getStd();
    vector<double> getRange();
};
```

- 建立payoff類宣告用蒙地卡羅法算各式選擇權時都會用到的共同功能
- getValues：獲得每次蒙地卡羅法算出的選擇權價格vector
- getMean：由選擇權價格vector的數字計算平均
- getStd：由選擇權價格vector的數字計算標準差
- getRange：查看選擇權價格vector中的最大與最小值

# VanillaOption

```
class VanillaOption : public Payoff{
private:
    Simulator* simulator;
    char whichOption;
    double K, r, T;

public:
    VanillaOption(Simulator *simulator, char whichOption, double k);
    virtual ~VanillaOption();

public:
    virtual void calc() override;
    virtual void deletePrice() override;
};
```

```
void VanillaOption::calc() {
    const array3d& totalPrice = simulator->getPrice();
    for(int i = 0; i < totalPrice.size(); ++i){
        double sumPayoff = 0;
        for(int j = 0; j < totalPrice[0].size(); ++j){
            if (whichOption == 'C'){
                sumPayoff += max<double>(totalPrice[i][j][totalPrice[0][0].size() - 1] - K, 0);
            } else if(whichOption == 'P'){
                sumPayoff += max<double>(K - totalPrice[i][j][totalPrice[0][0].size() - 1], 0);
            }
        }
        double avePayoff = sumPayoff / totalPrice[0].size();
        double discountedValue = avePayoff * exp(-r * T);
        optionValues.push_back(discountedValue);
    }
}
```

- 繼承payoff類的共同功能
- 重寫VanillaOption自己的payoff計算方式
- 計算方式：直接用最後一期的價格和履約價相減計算
- 計算多條模擬路徑的平均價格後折現

# LookBackOption

```
#include "Payoff.h"

class LookBack final : public Payoff {
private:
    Simulator* simulator;
    char whichOption;
public:
    void calc() override;

    void deletePrice() override;

public:
    LookBack(Simulator *simulator, char whichOption);

    virtual ~LookBack();
};
```

- 會接收simulaoctr的三維股價矩陣，藉由讀取股價生成Payoff的三維矩陣，在對個路徑折現並取平均求得選擇權價值。
- 其中又分為
  - 1. LookBackCall: Max股價 - 到期日股價
  - 2. LookBackPut: 到期日股價 - Min股價

# AsianOption

```
class AsianOption : public Payoff{
private:
    Simulator* simulator;
    char whichOption;
    double K, r, T;
public:
    AsianOption(Simulator *simulator, char whichOption, double k);
    virtual ~AsianOption();

public:
    virtual void calc() override;
    virtual void deletePrice() override;
};

void AsianOption::calc() {
    const array3d& totalPrice = simulator->getPrice();
    for(int i = 0; i < totalPrice.size(); ++i) {
        double sumPayoff = 0;
        for (int j = 0; j < totalPrice[0].size(); ++j) {
            double sumPrice = 0;
            for(int m = 0; m < totalPrice[0][0].size(); ++m){
                sumPrice += totalPrice[i][j][m];
            }
            double avePrice = sumPrice / totalPrice[0][0].size();
            if(whichOption == 'C'){
                sumPayoff += max<double>(avePrice - K, 0);
            } else if(whichOption == 'P'){
                sumPayoff += max<double>(K - avePrice, 0);
            }
        }
        double avePayoff = sumPayoff / totalPrice[0].size();
        double discountedValue = avePayoff * exp(-r * T);
        optionValues.push_back(discountedValue);
    }
}
```

- 繼承payoff類的共同功能
- 重寫AsianOption自己的payoff計算方式
- 計算方式：用最後一期算出的平均價格和履約價相減計算
- 計算多條模擬路徑的平均價格後折現



# CRRBinomialTree:

```
enum OptionType{E = 1,A};

class CRRBinomialTree {
private:
    // user input
    const double S;
    const double T;
    const int n;
    const double sigma;
    const double r;
    const double q;
    const double K;
    const char whichOption;
    OptionType type;

    // container
    vector<double> treeInOneVector;

    // parameter calculation
    double delta_T = T/ n;
    double u = exp(sigma * sqrt(delta_T));
    double d = 1 / u;
    double p = (exp((r - q) * delta_T) - d) / (u - d);

    // calculate one price
    inline double calcPrice(int time, int num_from_top);

    // calculate one call payoff at maturity
    inline double callPayoffAtMaturity(int num_from_top);
```

- 用於計算Plain Vainilla Option，採用Q measure下的二項機率來求解整個股價樹，最後利用One vector 的方式節省二項樹所需要的儲存空間，並再執行 backward reductio 求解出股價。

```

class Simulator {
private:
    array3d price;
    vector<double> reductionRV;
    double S, q, r, T, sigma;
    int Rep, Sim, TradeDays;

    // Normal RV
    double getNormal();

    // calculate price and put in array3d price
    void calc();

```

```

public:
    // Constructor and destructor
    Simulator(double S, double r, double q, double T, double sigma, int Rep, int Sim, int TradeDays);
    virtual ~Simulator();

    // getter
    const array3d &getPrice() const;

    double getS() const;

    double getQ() const;

    double getT() const;

    double getSigma() const;

    int getRep() const;

    int getSim() const;

    double getR() const;

    int getTradeDays() const;
};

```

### Simulator:

用於生成path-dependent simulation by BSM，可藉由參數傳入不同的Payoff裡面可以作為中間媒介生成選擇權價格

```

#include "Payoff.h"

class LookBack final : public Payoff {
private:
    Simulator* simulator;
    char whichOption;
public:
    void calc() override;

    void deletePrice() override;

public:
    LookBack(Simulator *simulator, char whichOption);

    virtual ~LookBack();
};

```

### LookBack:

會接收simulaoctr的三維股價矩陣，藉由讀取股價生成Payoff的三維矩陣，在對個路徑折現並取平均求得選擇權價值。其中又分為

1. LookBackCall:  $\text{Max股價} - \text{到期日股價}$
2. LookBackPut:  $\text{到期日股價} - \text{Min股價}$

```

enum OptionType{E = 1,A};

class CRRBinomialTree {
private:
    // user input
    const double S;
    const double T;
    const int n;
    const double sigma;
    const double r;
    const double q;
    const double K;
    const char whichOption;
    OptionType type;

    // container
    vector<double> treeInOneVector;

    // parameter calculation
    double delta_T = T / n;
    double u = exp(sigma * sqrt(delta_T));
    double d = 1 / u;
    double p = (exp((r - q) * delta_T) - d) / (u - d);

    // calculate one price
    inline double calcPrice(int time, int num_from_top);

    // calculate one call payoff at maturity
    inline double callPayoffAtMaturity(int num_from_top);

```

### CRRBinomialTree:

用於計算Plain Vanilla Option，採用Q measure下的二項機率來求解整個股價樹，最後利用 One vector 的方式節省二項樹所需要的儲存空間，並再執行 backward reductio 求解出股價。