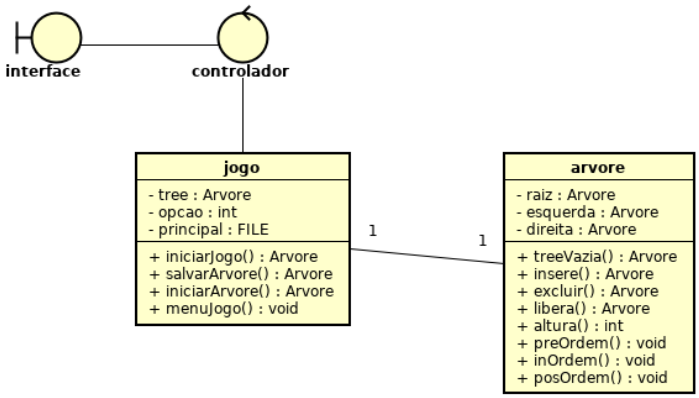
Check List Trabalho 3

Nome: Gabriel Porto Oliveira
180058975

2- Design do Software

1- Fazer diagrama de classes : Não passou.



2- Definir como as telas ou CLI serão organizadas: Passou.

3-Definir qual SGBD será utilizado: Não passou. Nenhum SGBD foi utilizado.

4- Escolher as linguagens necessárias de acordo com a necessidade: Passou.

5- Defnir a equipe necessária e quem vai desenvolver qual parte do software:Passou.

6- Dividir o software em funcionalidades e como serão desenvolvidas e implementadas:Passou.

7- Esquematizar todas as possíveis ações do usuário:Passou.

8- Formular soluções para todas as possíveis exceções:Passou.

9- Planejar todas as funções, parâmetros e como esses irão funcionar:Passou.

10- Fazer esboços de como a GUI ou CLI podem ser antes de implementar:Passou.

11-Utilizar mais diagramas de modelagem para organizar o máximo possível:Não passou. Programa é básico demais para necessitar outros diagramas.

12- Pensar em formas de otimizar a ordem de execução das funções para ter maior performance:Passou.

13- Decidir a forma de programação:Passou.

14-Estabeler metas a cada uma ou duas semanas: Passou.

15- Planejar o software de forma que seja fácil mante-lo mais tarde ou por outra equipe: Não passou. Mais comentários foram adicionados.

16- Criar padrões de qualidade a ser atingidos: Passou.

17- Montar objetos de acordo com os usuários do programa: Não passou. Não c ontém objetos.

18- Utilizar somente estruturas que são necessárias: Passou.

19- Levar em consideração as limitações físicas do hardware alvo: Não passou. Não foi levado em consideração o hardware alvo, mas como o programa é bem simples não é necessário.

20- Projetar funções que podem ser reutilizadas: Não passou. Tirando as funções base da árvore, as outras funções são únicas com apenas um uso.

3 – Padrões de programação do software.

1- Não utilizar nomes genêricos para as variáveis: Passou.

2- Utilizar nomes de variáveis que façam sentido: Passou.

3- Comentar o que as funções fazem: Passou.

4- Comentar partes complexas do código: Passou.

5- Criar funções com nomes auto-explicativos: Passou.

6- Identar codigo corretamente: Passou

7- Não criar código confuso: Não Passou. Algumas linhas e funções foram melhoradas para clareza.

8- Criar padrões de codificação: Passou.

9- Dividir grandes partes do código em funções independentes: Não passou. Algumas funções poderiam ser quebradas em outras.

10- Comentar as variáveis utilizadas: Não passou. Algumas variáveis não estavam comentadas.

11- Codifcar de maneira que seja fácil manter: Não passou.

12- Criar código que outros entendam: Passou.

13- Utilisar sistema de gerencimanto de versões: Passou.

14- Separar o código em vários arquivos: Passou.

15- Reutilizar funções o máximo possível: Não passou. Apenas as funções da árvore podem ser reutilisadas.

16- Usufruir das funcionalidades da liguagem: Passou.

17- Entender bem os conceitos da linguagem: Passou.

18- Não utilizar variáveis globais: Passou.

19- Entender como a alocação de memória ocorre: Passou.

20- Utilizar constantes invés de #define: Passou.

4- Padrões para testagem de Software.

1- Utilizar alguma biblioteca de testes de código: Passou.

2- Testar todas as funções existentes no código: Passou

3- Checar se os testes fazem sentido: Passou.

4- Utilizar todas as ferramentas disponibilizadas pelos frameworks de teste: Não passou. Poderia ter utilizado mais ferramentas.

5- Testar se todas as funções implementadas são utilizadas: Passou.

6- Checar se parâmetros fora do esperado quebram o software ou se ele consegue seguir funcionando normalmente: Passou.

7- Criar testes que levem em consideração as possíveis opções do usuário e exceções: Não passou. Poderia ter implementado vários teste para a mesma função.

8- Utilizar vários frameworks de teste simultaneamente: Não passou. Apenas um framework utilizado.

9- Testar utilizando input humano, como um colega de trabalho: Não passou. Apenas o desenvolvedor testou.

10- Testar para erros de sintaxe e lógica: Passou.

11- Utilizar diversos compiladores: Não passou. Foi utilizado apenas o G++.

12- Testar para usabilidade:Passou.

13- Testar as funções assim que elas são feitas: Passou.

14- Utilizar a ajuda de outros programadores: Passou.

15- Criar unidades de testes independentes do código principal: Passou.

16- Corrigir erros apontados pelos testes rapidamente e com soluções inteligentes: Passou.

17- Entender como os diferentes frameworks funcionam antes de utiliza-los.Passou.

18- Usar linux: Passou muito bem.

19- Testar utilizando vários tipos de variáveis: Passou.

20- Usar testes para entender o código: Passou.

5- Padrões para debugging de Software.

1- Usar debugger como o gdb: Passou.

2- Entender como o debugger funciona para otimizar sua utilização: Passou.

3- Utilizar GUI para facilitar o processo e ficar mais fácil de visualizar erros: Passou.

4- Utilizar alguma IDE, isso pode diminuir as chances de algo dar errado e integrar o desenvolvimento. Facilitando o processo de debugging. Passou.

5- Depurar funções independentes e separadamente: Passou.

6- Modifcar variáveis durante a depuração: Passou.

7- Visualizar a mudança dos valores em real-time: Passou.

8- Não depurar o programa inteiro de uma vez: Passou.

9- Não deixe de utilizar GUIs se elas te ajudam: Passou.

10- Preste atenção em ponteiros, uma pequena diferença pode quebrar o software: Passou.

11- Comentar códigos já depurados e testados: Passou.

12- Checar o comportamento linha por linha: Passou.

13- Não colocar breakpoints em linhas vazias: Passou.

14- Entender as mensagens de erro: Passou.

15- Comentar linhas de código: Passou.

16- Diminuir os breakpoints com o passar do tempo, assim é possível diminuir o número de possíveis linhas de código com problema: Passou.

17- Testar os arquivos .c ou .cpp um de cada vez: Passou.

18- Coloque breakpoints no inicio e fnal de laços para saber se o problema esta antes, depois ou dentro deles: Passou.

19- Utilizar breakpoints em ações com ponteiros: Passou.

20- Colocar breakpoints em chamadas de funções: Passou.