RUMMIKUB		
q1110490	**4**	Assignment 1: Note that instead of using the =:=/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X =:= Y. It is good practice to use Number is PrevNumber+1 instead of Number =:= PrevNumber+1 when only the RHS should be evaluated. Assignment 2: Be aware of the sublist/2 or sublist/3 predicate. The play_game/4 predicate should output actions of the players, you return whether or not the second player has won. Singleton variables are variables that have a name but are not used anywhere else in the predicate. Read the conditions for each case very carefully (e.g. to draw from the bag you cannot play a row or block). If you call member/2 and select/3 subsequently, member/2 becomes superfluous. In can_play_block/4 you want to sort the blocks instead of the row. You can use append(A,[B],C) to add an element to the end of a list. Instead of sublist you can also call select/3 three times.
q0986526	**6**	Assignment 1: valid_table/1 does not return the table, if you are still confused contact me. Note that instead of using the =/2 or =:=/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X =:= Y. Also, it is good practice to replace 'Number + 1 =:= N1' by 'N1 is Number +1'. Assignment 2: Replace 'length(B1,L), L is 0' by a pattern match on [ ]. If you play a row, you want to remove the selected blocks from the player's list of blocks: look at the select/3 predicate. By using findnsols/4 you might miss possible solutions, the findall/3 predicate is more appropriate. Use [_|_] instead of [_,_] to add an element to the front of a list. Instead of using reverse/2, which makes your solution a lot more complex, build up the Actions the other way around: draw/win/lose in a base case, and an action in the recursive case. Assignment 3: you assume the players have performed no actions, while -Actions is the output of the play_game predicate. In line 156 you need is/2 instead of =/2.
q1038722	**6**	Assignment 1: Note that an underscore before a variable (e.g. _Table) makes it an anonymous / don't care variable. You typically use this underscore when you have singleton variables: placeholder for variables that are not used anywhere else in the predicate. Assignment 2: Your draw/3 predicate does the same as unifying Newbag with [Block | OldBag]. Understand the difference between member/2 and select/3. If you use select/3, member/2 becomes superfluous. Be careful with [NewTable | NewRow], you are nesting lists --> I think you mean [NewRow | NewTable]. For `playrow` you should select 3 blocks from the player's blocks and check if they form a valid row together. Also, read the assignment very carefully to check the conditions for each case (e.g. a player can only draw from the bag if he can not play a block or a row). Thank you for the well-documented and structered code.
q1013358	**10**	Assignment 1: ok Assignment 2: ok Assignment 3: The predicate member(X,List) does the same thing as select(X,List, _). Thanks for the nice, readable code.
q1148275	**10**	Assignment 1: Typically, we use is/2 instead of =:=/2 if only the RHS should be evaluated. Assignment 2: Note that adding an element to the front of a list [Elem | List] is more performant than adding it to the end (append(List,[Elem], NewList)). It might be more elegant to build up your Actions the other way around, i.e. start with [win] or [lose] or [draw] in the base case and add actions to the front of the list in recursive steps. Similarly for adding rows to the table. pred(_) :- fail. does the same as leaving out this line.

RUMMIKUB		
q1031445	**4**	Assignment 1: You forget to check the length of a row, which should be at least 3. Replace D is X + 1, Y == D. by Y is X + 1. Assignment 2: Be aware of the select/3 and sort/2 predicates. To add an element to the front of a list, use [Elem | List]. Read the assignment very carefully to determine the conditions for each case (e.g. to draw from the bag, you should check that no other move is possible). Add base cases with win, lose or draw.
q0982362	**7**	Assignment 1: ok Assignment 2: Read the conditions from the assignment very carefully. A player only wins if his opponent has some blocks left. If both players ran out of blocks, you now match line 39, 40 and 41. You can assume the player in the first argument is on so that you don't need the extra argument in the helper predicate. You want to append a block to a row of blocks, not to nrow(Blocks) itself. Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. You are close, try to debug some of the test cases to find your bugs more quickly. Assignment 3: Your idea is fine, you have some mismatched brackets and need to pay attention to the order you put predicates in the findall. You can use member/2 instead of select/3 in this case.
q1148097	**5**	Assignment 1: Be aware of the length/2 predicate. Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. Assignment 2: Be aware of the difference between terms and predicates (playrow, playblock). Be aware of the select/3 predicate. You can add a block to the front or the end of a crow. You have to sort the crows to make the Esystant tests work (use sort/2). The append/3 predicates has as arguments three lists. To add a single element to a list use [Elem | List] to add it to the front or append(List, [Elem], NewList) to add it to the end of the list.
q1201686	**4**	Assignment 1: The valid_row/1 predicate is superfluous, it is just an other name for valid_table/1. Note that instead of using the =:=/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X =:= Y. Typically, we use is/2 instead of =:=/2 if only the RHS should be evaluated. Assignment 2: Indeed, playP1 and playP2 are the saem predicate. Nothe that the append/3 predicate takes three lists as arguments. You can either add an element to the front of a list [Elem|List] or to the end of the list, if necessary append(List,[Elem],NewList). Read the assignment very carefully to determine the conditions for each case (e.g. to draw from the bag, you should not be able to play a row or block). You might want to use select/3 instead of delete/3 and member/2 together. To assign a list to a variable you need '=' instead of 'is'. You accidentally call playCrow/2 in playNrow/2.
q1188628	**10**	Assignment 1: ok Assignment 2: If you assume the player in the first argument is on, you don't need the extra 'playerX' argument. Assignment 3: the flatten/2 predicate seems superfluous. Thank you for the nice, readable code!
q1111868	**6**	Assignment 1: Be aware of the select/3 predicate. Note that instead of using the ==/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X == Y. It might be more elegant to use \+/1 instead of member/2 and remove/3 on line 23-24. Assignment 2: Be aware of the sublist/2 predicate in prolog. You can add an element to the front of a list using [Elem | List], only add to the end of a row if really necessary. pred(_) :- fail is the same as leaving out this definition. Your code is not very readable, try to add some documentation to clarify your reasoning or use more intuitive names for predicates and variables. Assignment 3: using member/2 or last/2 might be helpful.

RUMMIKUB		
q1139593	**6**	Assignment 1: Note that instead of using the =/2 or is/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X is Y. The is/2 predicate is used when the RHS should be evaluated. Checking for number/1 and atom/1 is superfluous. Assignment 2: Determine the conditions for each case very carefully (e.g. only draw a block if no other move is possible). You create a loop by calling play_game_put/5 from play_game/4 and vice versa. Check the append/3 predicate, you may want to use it instead of select/3 on line 162 and others, or instead of calling reverse/2 twice.
q1133654	**4**	Assignment 1: Use [Elem | List] instead of append([Elem], List, NewList) to add an element to the front of the list. Typically, we use is/2 instead of =:=/2 if only the RHS should be evaluated. Assignment 2: Note that adding an element to the front of a list [Elem | List] is more performant than adding it to the end (append(List,[Elem],NewList)). It might be more elegant to build up your Actions the other way around, i.e. start with [win] or [lose] or [draw] in the base case and add actions to the front of the list in recursive steps. Similarly for adding rows to the table. You don't need to explicitly put quotes around win, lose, draw, you can see them as prolog atoms. The question concerning not(member(....)) is correct, you should indeed check this although it is not very performant. You might want to use select/3 instead of delete/3 together with member/2. Sort should transform the list into a sorted list without duplicates (compare to list_to_set/2). Assignment 3: This assignment could be solved in a more elegant way: use another findall/3 and member/2 to find the winning games. Be careful keeping the actions of both players, this makes your solution unnecessarily complex.
q1148273	**3**	Assignment 1: You have many singleton variables, which indicates you don't use them anywhere else and which might point to bugs. Read the warnings. You assume a row can only have length 3 but it can also be longer than that. Typically, we use is/2 instead of =:=/2 if only the RHS should be evaluated, so change X+1 =:=Y into Y is X+1. Assignment 2: Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred (X,Y) :- X = Y. You forgot to make a case for win-lose. You build up your list of actions in a strange way, you always add to the end of the list. Adding to the front of the list is more performant and clear ([Elem | List]). Be aware of the select/3 predicate. Drawing from the bag is not the same thing as ending the game with draw.
q1142025	**9**	Assignment 1: Replace line 38-39 by NumberJ is NumberI + 1. Assignment 2: You can replace line 57-58 by pattern matching on [_|_]. (Side note) It is more performant to add an element to the front of a list with [Elem | List] then to the end with append/3. To select 3 blocks, you can easily use select/3 three times, your block_subset_of_three_blocks/3 is overly complicated. Sorting a row after appending an element to the front or the end gives the same result. Overall, your program looks good, you only have small bugs or minor mistakes. Assignment 3: ok
q1124765	**8**	Assignment 1: ok Assignment 2: You can use append[List,[Elem],NewList) to add an element to the end of a list. If you assume the player in the first argument is on, you don't need the extra 'Turn' argument. Be careful with reversing the actions. Read the assignment very carefully to determine the conditions for each case (e.g. to draw from the bag you should check that no other move is possible). Assignment 3: You might write count_wins_in_list/3 in a more elegant way using findall/3.

RUMMIKUB		
q1428238	**7**	Assignment 1: ok. Assignment 2: Be aware of the select/3 predicate, you want to use it in triple_list_select/3. Read the assignment very carefully to determine the conditions for each case (e.g. only draw from the bag if no other move is possible).
q0982370	**3**	Assignment 1: Typically, we use is/2 instead of =:=/2 if only the RHS should be evaluated. Revisit the difference between =/2, ==/2, is/2 and =:=/2. Assignment 2: You made a nice decomposition of the problem. It is indeed important to make multiple defininitions of your predicates, one for each case. As you start implementing, you will see that even more helper predicates may be appropriate. Assignment 3: Decomposition looks ok, look at the findall/3 predicate, which may be more useful that foreach/2.
q1141634	**6**	Assignment 1: Indenting your predicates makes your solution more readable. Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. Assignment 2: Note that adding an element to the front of a list [Elem | List] is more performant than adding it to the end (append(List,[Elem],NewList)). You forget to remove the blocks to make a row (in play_row/4) from the player's list of blocks. Use the select/3 predicate for that. Read the assignment very carefully to determine the conditions for each case (e.g. only draw from the bag if no other move is possible). If you have warnings for singleton variables, that may point to a bug.
q1434056	**6**	Assignment 1: Be aware of the length/2 predicate. You forget to check the colours. Assignment 2: A player can only win when he has no blocks and his opponent does have one or more blocks left: Blocks can be either [ ] or [_|_]. Replace append([A],B,C) by C = [A|B] or by pattern matching C. When you create a row, you always take the first block as an element of that row, which is not always correct. Check whether the row you have created is valid or not. Assignment 3: ok
q0991124	**2**	Assignment 1: Instead of the functor/3 predicate, you can pattern match on the rows. Note that instead of using the =:=/2, ==/2, =/2 predicates, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. This may make your code more readable. Be aware of the built-in predicate sort/2. I think you lost time on implementing quicksort. You have several singleton variables, which are variables that appear once but are not used in your predicate. You can replace by an underscore of prefix an underscore to the variable (Number ==> _ or _Number). Note that in line 104 you put _Table, which is also an anonymous variable. Using _Table twice might not work correctly.
q1403912	**7**	Assignment 1: Instead of Len < 3,!, false, check that Len >= 3. Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. Assignment 2: In the base cases, add win-lose-draw to the player's actions. The remove_blocks/3 predicate is not necessary if you immediately use select/3 instead of member/2. Check whether the row is still valid when you played a block after the row is instantiated. Read the assignment very carefully to determine the conditions for each case (e.g. only draw from the bag if no other move is possible). Assignment 3: ok
q1125944	**9**	Assignment 1: Variables that start with an underscore are anonymous variables, if you use them more than once leave out the underscore. Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. Assignment 2: You forgot the case where both players have no blocks left Assignment 3: ok

RUMMIKUB		
q1276844	**1**	Assignment 1: You assume a table if of length 1 or 2 but it can be longer as well. Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. Avoid using the arg/3 predicate, instead pattern match on [Head | Tail] if you want to take the first element out of a list. The member/2 predicate is built-in. Assignment 2: I see that you were planning on decomposing the problem in multiple cases, which is a good idea. You can assume the player in the first argument of play_game is on, so you won't need the helper predicate with the extra argument. To play a row, select 3 blocks from the current player and check whether these 3 blocks together form a valid crow. Assignment 3: Familiarize yourself with the findall/3 predicate. You compute three times the same result. Note that div/2 floors the division, instead you want to use X / Y.
q0980053	**9**	Assignment 1: Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred (X,Y) :- X = Y. Assignment 2: There is a select/3 predicate that you might want to use instead of member/3 and delete/3. Due to the cuts, you only have one result per query. Instead, explicitly state the conditions for a certain case (e.g. for victory of player1, P1 should have no blocks [ ], P2 should have at least 1 block left: [_|_]). The 'Turn' argument is superfluous, you can assume the player in the first argument is on. Assignment 3: It is not explicitly stated that you should count draw as 0.5, that is why your Esystant tests do not work. The rest is ok but to write a more compact/elegant solution you may experiment a bit with the findall/3 predicate.
q1431267	**0**	Assignment 1: The head of the table list can be both an nrow or a crow. Make a helper predicate with an additional argument representing colours you have already seen.
q1135621	**4**	Assignment 1: Understand the difference between predicates and terms. Your crow/3 predicate can be simpler: write crow([block(N,C)|BL],N,L1):- \+ member(C,L1), crow(BL,N,[C|L1]). Similarly for nrow/3. Pay attention to the difference between =:=/2, is/2, ==/2 and =/2. Assignment 2: The draw/3 predicate is superfluous, you can just pattern match on [B|T] for your bag where you use it. You always take the first three elements from the list of blocks. You should also remember the remaining list of blocks. Use select/3 for that. You can add an element to the front of a list using [Elem | List], preferably use this over append/3. Provide base cases with win-lose-draw.
q1424842	**4**	Assignment 1: Try to give more intuitive names to your prolog terms. Check your base case, in line 25 it is incorrect. A singleton list can be written as [X] instead of [X | []]. Instead of using the ==/2 predicate, use the same symbol, e.g. pred (X,X) instead of pred(X,Y) :- X == Y. Your code for assignment 1 works but your style can improve (see previous remarks). Assignment2: You started with a good structure but try to work case by case, read the assignment very well (e.g. what are the conditions for having a `draw` for both players when the bag is empty?). Think about what are the inputs and outputs of your predicate. Write helper predicates.

RUMMIKUB		
q1431226	**7**	Assignment 1: Note that instead of using the ==/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X == Y. Assignment 2: Use the select/3 predicate instead of selectchk/3. Be aware of the difference between terms, predicates and facts. nrow/1 and crow/1 are used as terms, you also made predicates of those, making things confusing. In 'playblock' you seem to add the block twice to the row. Also, be careful with singleton variables (e. g. Rest, line 90), when you compile you get warnings of these. In your action/3 predicate it might be easier and more readable to have 7 arguments instead of putting 3 arguments (blocks, table, bag) together in a list. The draw(null) is confusing, try to avoid the use of 'null'. Assignment 3: The percentage is expressed as a value between 0 and 1.
q0682466	**9**	Assignment 1: ok Assignment 2: You miss a base case, where player 1 wins. Note that instead of using the =/2 predicate, you can use the same symbol, e.g. pred(X,X) instead of pred(X,Y) :- X = Y. You can add multiple elements to the front of a list using [Elem1, Elem2 | List]. You can replace line 125-127 by member(block(V,_C),Blocks). Assignment 3: ok
q1149197	**6**	Assignment 1: You can add two (or more) elements to the front of a list using [Elem1, Elem2 | List]. Note that instead of using the =:=/2 predicate, you can use the same symbol (when no computation), e.g. pred(X,X) instead of pred(X,Y) :- X =:= Y. Assignment 2: You can write a singleton list as [draw] instead of [draw|[]], similar for three elements. You can assume the player in the first argument is on so you don't need the extra argument. Read the assignment very carefully to determine the correct conditions for each case (e.g. only draw from the bag if no other move is possible). Note that you have to sort the elements in a row to make the tests work.
q1421139	**4**	Assignment 1: Pay attention to the difference between is/2, =/2 and ==/2, you use them interchangeably. Most of the time you can just pattern match instead (e.g. pred(X,X) instead of pred(X,Y) :- X =Y). It is good practice to instantiate all arguments when you call a helper predicate. Assignment 2: Read the assignment very carefully to have all conditions for each case (e.g. player1 only wins if he has no blocks and player2 still has blocks). There is a built-in predicate sort/2, so implementing bubble sort is not necessary. Use capitals to refer to variables (line 55,56,82,88). What you want to do in play_nrow_helper/2 is select 3 blocks and remember both these 3 blocks AND the list of remaining blocks for the current player (similar for play_crow_helper). You now check whether the number of blocks the current player has, equals 3.
q1106863	**8**	Assignment 1: In line 6-8 you reconstructed Table. Note that instead of using the is/2 predicate, you can use the same symbol (when no computation), e.g. pred(X,X) instead of pred(X,Y) :- X is Y. You can add an element to the front of a list using [Elem | List], which is more performant and readable than appending it to the end of a list using append/3. Assignment 2: You miss a base case where both players have no blocks left. Be aware of the select/3 predicate. Assignment 3: ok
q1152490	**6**	Assignment 1: ok Please provide a .pl file next time, I could not provide extensive feedback because your file was hard to read. Assignment 2: Read the conditions for each case very carefully in the assignment. You can pattern match on a non-empty list using [_|_].

RUMMIKUB	
q0985203 **7**	Assignment 1: You assume there are only four colors, which is correct for Rummikub but try to also think of a more general way to check if all colors are different. In general, it might be helpful to use the length/2 and findall/3 predicates more frequently. Assignment 2: the structure of your code looks good. You wrote helper predicates for smaller tasks. However, read the assignment very carefully to know the conditions for each case (e.g. the players only get a 'draw' if the bag is empty if they both still have blocks --> match their blocks with [_|_]). Be aware that, when both player's blocks are empty, they match the case in line 40 AND in line 41, leading to too many solutions. Similarly, the conditions to draw from the bag is that player 1 can not make a row or add a block to the table. Note that you can also add a block to the end of a row on the table. Assignment 3: Looks correct, but again you might get a more elegant solution using the findall/3 predicate.